Courcelle's Theorem – A Game-Theoretic Approach[☆]

Joachim Kneis, Alexander Langer, Peter Rossmanith

Department of Computer Science, RWTH Aachen University

Abstract

Courcelle's Theorem states that every problem definable in Monadic Second-Order logic can be solved in linear time on structures of bounded treewidth, for example, by constructing a tree automaton that recognizes or rejects a tree decomposition of the structure. Existing, optimized software like the MONA tool can be used to build the corresponding tree automata, which for bounded treewidth are of constant size. Unfortunately, the constants involved can become extremely large – every quantifier alternation requires a power set construction for the automaton. Here, the required space can become a problem in practical applications.

In this paper, we present a novel, direct approach based on model checking games, which avoids the expensive power set construction. Experiments with an implementation are promising, and we can solve problems on graphs where the automata-theoretic approach fails in practice.

Courcelle's celebrated theorem essentially states that every problem definable in Monadic Second-Order logic (MSO) can be solved in linear time on graphs of bounded treewidth [1]. However, the multiplicative constants in the running time, which depend on the treewidth and the MSO-formula, can be extremely large [2].

Theorem 1 ([1, 2])**.** *Let $\mathcal{P}$ be an MSO problem and w be a positive integer. There is an algorithm A and a function $f\colon \mathbf{N}\times\mathbf{N}\to\mathbf{N}$ such that for every graph $\mathcal{G}=(V,E)$ of order $n:=|V|$ and treewidth at most w, A solves $\mathcal{P}$ on input $\mathcal{G}$ in time $f(\|\varphi\|,w)\cdot n$, where φ is the MSO formula defining $\mathcal{P}$ and $\|\varphi\|$ is its length. Furthermore, unless* P = NP*, the function f cannot be upper bounded by an iterated exponential of bounded height in terms of φ and w.*

This result has been generalized by Arnborg, Lagergren, and Seese to Extended MSO [3], and by Courcelle and Mosbah to Monadic Second-Order evaluations using semiring homomorphisms [4]. In both cases, an MSO-formula with

[☆]Supported by the DFG under grant RO 927/8

Email addresses: kneis@cs.rwth-aachen.de (Joachim Kneis), langer@cs.rwth-aachen.de (Alexander Langer), rossmani@cs.rwth-aachen.de (Peter Rossmanith)

free set variables is used to describe a property, and satisfying assignments to these set variables are evaluated in an appropriate way.

Courcelle's Theorem is usually proved as follows: In time only dependent on φ and the treewidth w, a tree automaton $\mathcal{A}$ is constructed that accepts a tree decomposition of width w if and only if the corresponding graph satisfies the formula. This construction can either be done explicitly, by actually constructing the tree automaton (see, e.g., [3, 5, 6, 7, 8, 9, 10]), or implicitly via auxiliary formulas obtained by applying the Feferman–Vaught Theorem [11] extended to MSO [1, 12] (see, e.g., [1, 13, 14, 15, 10]).

In a practical setting, the biggest strength of Courcelle's Theorem is at the same time its largest weakness: MSO logic has extremely large expressive power, and very short formulas can be used to encode NP-hard problems. This is used in [2] to prove non-elementary worst-case lower bounds for the multiplicative constants in the linear running time. Even worse, these lower bounds already hold for the class of trees, i.e., graphs of treewidth one.

On the other hand, these are worst-case lower bounds for very special classes of formulas and trees, and thus there is a good chance that in practice problems can be solved much faster. In fact, existing software like the MONA tool [16, 17] for Weak Second-Order logic on two successors (WS2S) is surprisingly successful even though it is subject to the same theoretical lower bounds.

The automata-theoretic approach is therefore a promising starting point for practical applications of Courcelle's Theorem, particularly since advanced and optimized tools like MONA can be used as a black box for the majority of the work, and techniques like minimizing tree automata are very well understood.

There are, however, some cases where the automata-theoretic approach is infeasible in practice, i.e., when the automata (or set of auxiliary formulas) are too large to be practically computable. This can even happen when the final minimal automata are small, but intermediate automata cannot be constructed in reasonable time and space (note that each quantifier alternation requires an automaton power set construction).

In his thesis [18], Soguet has studied the sizes of tree automata corresponding to various problems for small clique-width [19].[1] The automata were generated using MONA, and in many cases, the corresponding automata were surprisingly small, thanks to the well-understood minimization of tree automata. On the other hand, even for graphs of clique-width three, MONA was unable to construct the corresponding tree automata for the classical 3-Colorability problem. Even worse, the same happened for simple problems such as deciding whether the graph is connected or if its maximum degree is two.

These negative results are somewhat unsatisfying because the respective algorithm already fails in the first phase, when the automaton is constructed.

[1] Both, treewidth and clique-width, can be defined in terms of graph grammars (hyperedge replacement grammars for treewidth, and vertex replacement grammars for clique-width; see e.g. a recent survey [20]), and in both cases, tree automata can be used to recognize parse trees of graphs.

The first phase, however, only depends on the treewidth (or clique-width in above cases) and the formula (i.e., the problem), but is independent of the actual input graph. On the other hand, when running the tree automaton on most graphs arising from practical problems, only few states are actually visited.

Recently, there have been a few approaches to this problem, see, e.g., [21, 22, 23, 24]. For example, the approach of [23, 24] avoids an explicit construction of the tree automaton. Instead, the state-transition function is computed *on-the-fly*. Experiments indicate practical feasibility. Courcelle [25] introduces *special tree-width*, where the corresponding automata are easier to construct.

In this paper, we present a novel, game-theoretic approach, where the input structure is taken into account from the beginning via model checking games (cf., [26, 27, 28]). Therefore, only the amount of information is stored that is needed by the algorithm to solve the problem on this explicit input, and, in some sense, transitions between nodes of the tree decompositions are as well computed *on-the-fly*. We particularly avoid the expensive power set construction.

We hope that the approach can be used in those cases, where the automata are too large to be constructed in practice, but the input graphs itself are simple enough. In fact, first experiments are promising. Using the generic approach, we can, for example, solve the 3-Colorability problem on grids of size 6×33 (treewidth 6) in about 21 seconds and with 8 MB memory usage on standard PC hardware, and the Minimum Vertex Cover problem on the same graph in less than a second and only 1 MB of memory usage. We note that the automata construction using MONA in [18] already failed for $2 \times n$ grids (clique-width 3).

Related Work

We briefly survey other approaches to Courcelle's Theorem. We already mentioned that, given the MSO formula φ, one can construct a finite-state bottom-up tree automaton that accepts a tree decomposition of the input graph G if and only if $G \models \varphi$. This is sometimes called the automata theoretic approach. A direct construction of the tree automata is described in, e.g., [9] or [10, Chapter 6]. In [29, 6] a Myhill-Nerode type argument is used to show that the treewidth parse tree operators admit a right congruence with finitely many congruence classes. The method of test sets can then be used to construct the tree automaton. One can also use a reduction to the classical model checking problem for MSO on labeled trees [3, 7, 8]. It is well-known [30, 31] that this problem can be solved by constructing suitable finite-state tree automata. This approach is favorable if one likes to use existing software such as the MONA tool [16].

A model theoretic approach is based on variants of the Feferman–Vaught Theorem [11]: If a graph G can be decomposed into components G_1 and G_2, then from the input formula φ one can construct a suitable *reduction sequence* consisting of Boolean combinations (*and*, *or*, *not*) of finitely many formulas that hold in G_1 and G_2 if and only if φ holds in G (cf., [1, 12, 14, 10]). One can therefore use dynamic programming on the tree decomposition to compute the *q-theory* of G, i.e., set of formulas of quantifier rank at most q that hold in G (cf., [13, 15, 14]). Similarly, one can also inductively compute the set of satisfying assignments to the input formula [4].

We are not aware of any implementations of Courcelle's Theorem based on the Feferman–Vaught approach. The construction of all possible reduction sequences for MSO formulas "obviously is not practical" [14, Section 1.6]. The algorithms presented in [13, 14] are therefore infeasible in practice. However, from [4] we get that computing the particular reduction sequence for the input formula φ suffices. Some lower bounds are known for the necessary conversions into disjunctions [32], but it would still be interesting to see how this approach behaves in practice.

A few authors studied practical aspects of the automata theoretic approach. It is mentioned in [6] that a Myhill–Nerode based program has been implemented as part of an M.Sc. thesis, which unfortunately does not seem to be publicly available. The MONA tool [16] is a well-known and optimized implementation for the tree automata construction. The space required to construct the automata with MONA still turns out to cause severe problems in practical applications [18, 22]. One idea [10, Chapter 6] is to use precomputed automata for commonly used predicates such as $Conn(X)$ expressing that the set X is connected. Note however that the $Conn(X)$ automaton requires $2^{2^{\Theta(k)}}$ states for graphs of clique-width k [10, Chapter 6]. An automatic translation into Monadic Datalog is proposed in [22]. Some experiments indeed suggest feasibility in practice; their prototype implementation was, however, obtained by manual construction and not by an automatic transformation from the underlying MSO formula. In [23, 24] the power set construction is avoided by considering existential formulas only. The automata thus remain non-deterministic, but of course standard methods to simulate runs of the automata apply. Since the state transition function is given only implicitly, the automaton is essentially computed *on-the-fly* while recognizing a clique-decomposition. Experiments have been conducted on graphs of comparably high clique-width and the approach is quite promising. In fact, the lack of feasible algorithms to compute the necessary clique-width parse trees seems to be the major limitation. To ease the specification of such *fly-automata*, Courcelle [25, 33] introduces *special tree-width*. Special tree-width lies between path-width and treewidth, but the automata are significantly smaller and easier to construct than those for treewidth.

In this article, we present a new approach that neither uses automata theoretic methods nor uses a Feferman–Vaught style splitting theorem. Instead, we essentially evaluate the input formula on the graph using a simple recursive model checking algorithm. In what follows, we shall outline this approach.

Overview

Our starting point is the *model checking game* for MSO (Definition 3), a pebble game between two players called the *verifier* and the *falsifier* also known as the Hintikka game [26]. The verifier tries to prove that the formula holds on the input structure, while the falsifier tries to prove the opposite. In the game, the verifier moves on existential formulas ($\vee$, $\exists$), while the falsifier moves on universal formulas ($\wedge$, $\forall$).

This game can in a natural way be identified with a simple algorithm that *evaluates* the formula on the input structure in a recursive manner. If, for example, the formula is $\exists R\psi(R)$ for a set variable R, the algorithm checks whether $\psi(U)$ holds for all sets U. In this sense, the computation tree of this simple algorithm can be interpreted as the *unfolding* (cf., [34]) of the model checking game. On a structure with n elements, this straight-forward recursive model-checking algorithm takes time $O((2^n + n)^q)$ for a formula of quantifier rank q. By dynamic programming on the tree decomposition, we can improve this to time linear in n on structures of bounded treewidth.

This works as follows: We traverse the tree decomposition of the input structure $\mathscr{A}$ bottom-up. At each node of the tree decomposition we preliminary try to evaluate the formula φ on $\mathscr{A}$ using the model checking game on the "current" substructure $\mathscr{A}'$ of $\mathscr{A}$. To this end, we allow "empty" assignments $x := \text{nil}$ to first order variables x. Such empty assignments correspond to objects in $\mathscr{A}$ that are not contained in $\mathscr{A}'$ and are to be assigned in later steps. Then, two things may happen:

- We can *already now* determine whether $\mathscr{A} \models \varphi$ or $\mathscr{A} \not\models \varphi$.

  If, for instance, the formula *3col* encodes the 3-Colorability problem and even $\mathscr{A}'$ is not three-colorable, it locally violates *3col* and we can derive $\mathscr{A} \not\models$ *3col*.

- We cannot *yet* determine whether $\mathscr{A} \models \varphi$ or $\mathscr{A} \not\models \varphi$.

  For example, if the formula encodes Dominating Set problem, then a vertex v in the "current" bag might be undominated in the current subgraph, but we do not know whether in the "future" another vertex might dominate v.

The first case is formalized in Lemma 4 and Lemma 6. In the second case, we found a "witness," i.e., a subgame that we were unable to evaluate. We then will re-visit those undetermined subgames during the course of the dynamic programming until we finally arrive in the root of the tree decomposition, where all subgames become determined.

The next crucial observation is that MSO and FO formulas with bounded quantifier rank have limited capabilities to distinguish structures (formally captured in the $\equiv_q$-equivalence of structures, cf. [35]). We exploit this fact and show that we can delete redundant equivalent subgames (cf., Algorithm 3) for a suitable definition of equivalence (cf., Definition 5). We can then show that, assuming a fixed formula and bounded treewidth, the number of reduced, non-equivalent games is bounded by a constant (Lemma 8), which allows us to obtain running times linear in the size of the tree decomposition.

While this game-theoretic approach is subject to the same non-elementary lower bounds as the other approaches, the actual number of ways to play the model checking game highly depends on the input graph. For example, if the graph does not contain, say, a triangle, then the players will never move to a set of nodes that induce a triangle, while a tree automaton must work for all

graphs. This observation is reflected in practical experiments, where the actual number of entries considered is typically much smaller than the corresponding worst-case bound.

1. Preliminaries

The power set of a set U is denoted by $\mathscr{P}(U)$. The disjoint union of two sets U_1, U_2 is denoted by $U_1 \uplus U_2$. We assume that trees are rooted and denote the root of a tree $\mathcal{T}$ by $root(\mathcal{T})$. For every $t \in \mathbf{N}$, $\exp^t(\cdot)$ is a t-times iterated exponential, i.e., $\exp^0(x) = x$ and $\exp^t(x) = 2^{\exp^{t-1}(x)}$.

For a set U and object x, we let $(x \in U)$ be defined as

$$(x \in U) = \begin{cases} 0 & \text{if } x \notin U \\ 1 & \text{if } x \in U. \end{cases}$$

To avoid cluttered notation, we may, for elements $s_1, \ldots, s_l$ and $t_1, \ldots, t_m$, abbreviate $\bar{s} := \{s_1, \ldots, s_l\}$, $(\bar{s}, s') := \bar{s} \cup \{s'\}$, and $(\bar{s}, \bar{t}) := \bar{s}\bar{t} := \bar{s} \cup \bar{t}$.

1.1. Structures

We fix a countably infinite set of *symbols*. Each symbol S has an *arity* $r = arity(S) \geq 0$. We distinguish between *nullary* symbols with arity zero and *relation symbols* that have arity greater than zero. Relation symbols with arity one are called *unary*. For convenience, we shall denote relation symbols by capital letters and nullary symbols by lower case letters.

A *vocabulary* τ is a finite set of symbols. We denote by $null(\tau)$ the set of nullary symbols in τ, by $rel(\tau)$ the set of relation symbols in τ, and by $unary(\tau)$ the set of unary relation symbols in τ. Let $arity(\tau) = \max\{\, arity(R) \mid R \in rel(\tau) \,\}$ be the maximum arity over all relation symbols in τ. If $null(\tau) = \emptyset$, we call τ *relational*.

Let τ be a vocabulary. A *structure* $\mathscr{A}$ *over* τ (or τ-structure) is a tuple $\mathscr{A} = \big(A, (R^{\mathscr{A}})_{R \in rel(\tau)}, (c^{\mathscr{A}})_{c \in null(\tau)}\big)$, where A is a finite set called the *universe* of $\mathscr{A}$, and $(R^{\mathscr{A}})_{R \in rel(\tau)}$ and $(c^{\mathscr{A}})_{c \in null(\tau)}$ are *interpretations* of the τ-symbols in $\mathscr{A}$. Here, $R^{\mathscr{A}} \subseteq A^{arity(R)}$ for each relation symbol $R \in rel(\tau)$. For a nullary symbol $c \in null(c)$ we either have $c^{\mathscr{A}} \in A$ and say that c is *interpreted* in $\mathscr{A}$, or we write $c^{\mathscr{A}} = \text{nil}$ and say that c is *uninterpreted*. The set of nullary symbols interpreted in $\mathscr{A}$ is denoted by $interpreted(\mathscr{A})$. If all symbols are interpreted, we say the structure is *fully interpreted*, and *partially interpreted* otherwise. We note that a related concept of *partially equipped signatures* has been used in, e.g., [29, 6, 36].

The set of all τ-structures is denoted by $\mathcal{STR}(\tau)$. We shall always denote structures in script letters $\mathscr{A}, \mathscr{B}, \ldots$ and in roman letters $A, B, \ldots$ their corresponding universes. If the universe is empty, then we say that the structure is *empty*. Structures over a relational vocabulary τ are called *relational structures*.

For a structure $\mathscr{A}$, we denote by $vocabulary(\mathscr{A})$ the vocabulary of $\mathscr{A}$. For sets $\bar{R} = \{R_1, \ldots, R_l\} \subseteq rel(\tau)$ and $\bar{c} = \{c_1, \ldots, c_m\} \subseteq null(\tau)$, we let

$\bar{R}^{\mathscr{A}} := \{ R^{\mathscr{A}} \mid R \in \bar{R} \}$, and $\bar{c}^{\mathscr{A}} := \{ c^{\mathscr{A}} \mid c \in \bar{c} \cap \mathit{interpreted}(\mathscr{A}) \}$ be their corresponding interpretations.

Example 1. *A graph (V, E) can in a natural way be identified with a structure $\mathscr{G}$ over the vocabulary $\tau_{Graph} = (adj)$, where adj represents the binary adjacency relation. The universe of $\mathscr{G}$ is V, and we interpret adj as $adj^{\mathscr{G}} = E$ in $\mathscr{G}$.*

Let τ be a vocabulary and $\{R_1, \ldots, R_l, c_1, \ldots, c_m\}$ be a set of symbols, each of which is not contained in τ. The vocabulary $\tau' = (\tau, R_1, \ldots, R_l, c_1, \ldots, c_m)$ is called an *expansion* of τ. Similarly, if $\mathscr{A}$ is a τ-structure and $\mathscr{A}'$ is a τ'-structure that agrees with $\mathscr{A}$ on τ, i.e., $R^{\mathscr{A}} = R^{\mathscr{A}'}$ for each $R \in \mathit{rel}(\tau)$ and $c^{\mathscr{A}} = c^{\mathscr{A}'}$ for each $c \in \mathit{null}(\tau)$, then we call $\mathscr{A}'$ a τ'-expansion of $\mathscr{A}$. If $\mathscr{A}$ is a τ-structure, and $U_1, \ldots, U_l$ are relations over A, such that $U_i \subseteq A^{\mathit{arity}(R_i)}$, $1 \leq i \leq l$, and $u_1, \ldots, u_m \in A \cup \{\text{nil}\}$, we write $\mathscr{A}' = (\mathscr{A}, U_1, \ldots, U_l, u_1, \ldots, u_m)$ to indicate that $\mathscr{A}'$ is a τ'-expansion of $\mathscr{A}$, such that $R_i^{\mathscr{A}'} = U_i$, $1 \leq i \leq l$, and $c_j^{\mathscr{A}'} = u_j$, $1 \leq j \leq m$.

Let $\mathscr{A}$ be a τ-structure and $\bar{a} = \{a_1, \ldots, a_m\} \subseteq A$. Then $\mathscr{A}[\bar{a}]$ is the *substructure of $\mathscr{A}$ induced by $\bar{a}$*, where $\mathscr{A}[\bar{a}]$ has universe $\bar{a}$, for each relation symbol $R \in \tau$ we have $R^{\mathscr{A}[\bar{a}]} = R^{\mathscr{A}} \cap \bar{a}^{\mathit{arity}(R)}$, and nullary symbols c are interpreted as $c^{\mathscr{A}[\bar{a}]} = c^{\mathscr{A}}$ if $c^{\mathscr{A}} \in \bar{a}$ and become uninterpreted otherwise.

Two τ-structures $\mathscr{A}$ and $\mathscr{B}$ over the same vocabulary τ are *isomorphic*, denoted by $\mathscr{A} \cong \mathscr{B}$, if there is an *isomorphism* $h\colon A \to B$, where h is a bijection between A and B and

- $c \in \mathit{interpreted}(\mathscr{A})$ if and only if $c \in \mathit{interpreted}(\mathscr{B})$ for all $c \in \mathit{null}(\tau)$,
- $h(c^{\mathscr{A}}) = c^{\mathscr{B}}$ for every nullary symbol $c \in \mathit{interpreted}(\tau)$, and
- for every relation symbol $R \in \tau$ and $a_1, \ldots, a_p \in A$, where $p = \mathit{arity}(R)$,

$$(a_1, \ldots, a_p) \in R^{\mathscr{A}} \qquad \text{iff} \qquad (h(a_1), \ldots, h(a_p)) \in R^{\mathscr{B}}.$$

Definition 1 (Compatibility, Union)**.** We call two τ-structures $\mathscr{A}_1$ and $\mathscr{A}_2$ *compatible*, if for all nullary symbols $c \in \mathit{interpreted}(\mathscr{A}_1) \cap \mathit{interpreted}(\mathscr{A}_2)$ we have $c^{\mathscr{A}_1} = c^{\mathscr{A}_2}$ and the identity $x \mapsto x$ is an isomorphism between $\mathscr{A}_1[A_1 \cap A_2]$ and $\mathscr{A}_2[A_1 \cap A_2]$.

In this case, we define the *union* of $\mathscr{A}_1$ and $\mathscr{A}_2$, denoted by $\mathscr{A}_1 \cup \mathscr{A}_2$, as the τ-structure with universe $A := A_1 \cup A_2$ and interpretations $R^{\mathscr{A}_1 \cup \mathscr{A}_2} := R^{\mathscr{A}_1} \cup R^{\mathscr{A}_2}$ for every relation symbol $R \in \tau$. Nullary symbols $c \in \mathit{null}(\tau)$ with $c^{\mathscr{A}_1} = c^{\mathscr{A}_2} = \text{nil}$ remain uninterpreted in $\mathscr{A}_1 \cup \mathscr{A}_2$; otherwise $c^{\mathscr{A}_1 \cup \mathscr{A}_2} = c^{\mathscr{A}_i}$ if $c \in \mathit{interpreted}(\mathscr{A}_i)$ for some $i \in \{1, 2\}$.

1.2. Treewidth and Tree Decompositions

Tree decompositions and treewidth were introduced by Robertson and Seymour [37] in their works on the Graph Minors Project, cf. [6, 8, 38].

A *tree decomposition* of a relational τ-structure $\mathscr{A}$ is a tuple $(\mathcal{T}, \mathcal{X})$, where $\mathcal{T} = (T, F)$ is a rooted tree and $\mathcal{X} = (X_i)_{i \in T}$ is a collection of subsets $X_i \subseteq A$, such that

- $\bigcup_{i \in T} X_i = A$,
- for all p-ary relation symbols $R \in \tau$ and all $(a_1, \ldots, a_p) \in R^{\mathscr{A}}$, there is an $i \in T$ such that $\{a_1, \ldots, a_p\} \subseteq X_i$, and
- for all $i, j_1, j_2 \in T$, if i is on the path between j_1 and j_2 in $\mathcal{T}$, then $X_{j_1} \cap X_{j_2} \subseteq X_i$.

The sets X_i are called *bags*. The *width* of a tree decomposition is the size of its largest bag minus one, and the *treewidth* of a structure $\mathscr{A}$ is the minimum width of all tree decompositions of $\mathscr{A}$.

Without loss of generality, we assume that each tree decomposition we consider is *nice*. Nice tree decompositions are directed, where each edge in F has a direction away from the root, and have the following properties: Each node $i \in T$ has at most two children. For leafs $i \in T$, we have $X_i = \emptyset$. If i has exactly one child j, then there is $a \in A$ such that either $X_i = X_j \cup \{a\}$ or $X_i = X_j \setminus \{a\}$. In the former case, we say i is an *introduce* node, in the latter case we call i a *forget* node of the tree decomposition. Finally, if a node i has two children j_1 and j_2, then we require $X_i = X_{j_1} = X_{j_2}$ and call such nodes *join* nodes. If $i \to \cdots \to j$ is a directed path in $\mathcal{T}$ pointing away from the root, we say j appears *below* i in $\mathcal{T}$.

With every node $i \in T$ of a (nice) tree decomposition of a τ-structure $\mathscr{A}$ we associate a substructure $\mathscr{A}_i$ defined as follows: Let $A_i \subseteq A$ be the set of objects in X_i or in bags X_j for nodes j below i in the tree decomposition. Then we let $\mathscr{A}_i := \mathscr{A}[A_i]$ be the substructure of $\mathscr{A}$ induced by A_i.

Computing the treewidth of a graph is NP-complete [39]. However, the algorithms in this paper rely on a given tree decomposition of the input structure. For graphs G, there is a fixed-parameter tractable algorithm [40, 6] with a running time of $2^{O(tw(G)^3)}|G|$, whose dependence on the treewidth might become a problem in practical applications. In a practical setting, heuristics seem to work well and often nearly optimal tree decompositions can be computed [41]. Using *Gaifman graphs*, one can also compute tree decompositions of arbitrary structures, cf., [8, Section 11.3]. In the following, we therefore just assume a tree decomposition is given as part of the input. For more information on treewidth, we refer the reader to surveys such as [42, 43].

1.3. MSO Logic

MSO logic over a vocabulary τ, denoted by $\mathrm{MSO}(\tau)$, is simultaneously defined over all vocabularies τ by induction. Firstly, for every p-ary relation symbol $R \in \tau$ and any nullary symbols $c_1, \ldots, c_p \in \tau$, $\mathrm{MSO}(\tau)$ contains the *atomic* formula $R(c_1, \ldots, c_p)$. If R is unary, we may abbreviate $R(c)$ as $c \in R$. Secondly:

- If φ, ψ are in $\mathrm{MSO}(\tau)$, then $\neg\varphi$, $\varphi \vee \psi$, and $\varphi \wedge \psi$ are in $\mathrm{MSO}(\tau)$,
- If $\varphi \in \mathrm{MSO}(\tau \cup \{c\})$ for some nullary symbol c, then both, $\forall c\varphi$ and $\exists c\varphi$ are in $\mathrm{MSO}(\tau)$. This is called *first order* or *object quantification*.

- If $\varphi \in \mathrm{MSO}(\tau \cup \{R\})$ for a unary relation symbol R, then both, $\forall R\varphi$ and $\exists R\varphi$ are in $\mathrm{MSO}(\tau)$. The corresponding case is called *second order* or *set quantification*.

Note that we do not distinguish between "basic" symbols (contained in a certain "base" vocabulary such as τ_{Graph}), and symbols that are used as *variables* subject to quantification. Let τ be a vocabulary and $\varphi \in \mathrm{MSO}(\tau)$ be a formula. Let $\tau' \subseteq \tau$ be the smallest vocabulary with $\varphi \in \mathrm{MSO}(\tau')$. Then we call the symbols in $\mathit{unary}(\tau') \cup \mathit{null}(\tau')$ the *free* symbols of φ. Let $\|\varphi\|$ be the size of a suitable encoding of φ.

If $\varphi \in \{\forall c\psi, \forall R\psi, \psi_1 \wedge \psi_2\}$ for some c, R, ψ, ψ_1, and ψ_2, we call φ *universal*. Similarly, we call φ *existential* if $\varphi \in \{\exists c\psi, \exists R\psi, \psi_1 \vee \psi_2\}$.

If φ does not contain set quantifiers, then we say φ is *first order* and contained in $\mathrm{FO}(\tau)$. Note that in particular all atomic formulas of $\mathrm{MSO}(\tau)$ are first order. The *quantifier rank* $qr(\varphi)$ of a formula $\varphi \in \mathrm{MSO}(\tau)$ denotes the maximum number of nested quantifiers in φ, counting both first order and second order quantifiers, and is defined by induction over the structure of φ as

- $qr(\varphi) = 0$ if φ is an atomic formula,
- $qr(\varphi) = qr(\neg\varphi)$,
- $qr(\varphi) = \max\{qr(\psi_1), qr(\psi_2)\}$ if $\varphi \in \{\psi_1 \wedge \psi_2, \psi_1 \vee \psi_2\}$, and
- $qr(\varphi) = qr(\psi) + 1$ if $\varphi \in \{\forall R\psi, \exists R\psi, \forall c\psi, \exists c\psi\}$.

Without loss of generality, we assume throughout the paper that every formula is in negation normal form, i.e., the negation symbol $\neg$ only occurs in front of atomic formulas. This can be achieved by a simple rewriting of the formula.

For a fully interpreted τ-structure $\mathscr{A}$ and a formula $\varphi \in \mathrm{MSO}(\tau)$, we write $\mathscr{A} \models \varphi$ if and only if φ *holds* in $\mathscr{A}$ or is *true* in $\mathscr{A}$ in the classical sense, cf. [44, 35]. We shall do not specify this further, since we will switch to a game-theoretic characterization in the remainder of this paper, cf., Section 2.

In [3], Extended MSO was introduced. Here, an MSO-formula over a relational vocabulary is given together with an evaluation or optimization goal over the unary relation symbols (set variables). This principle was furthermore generalized to semiring homomorphisms in [4], where satisfying interpretations of the free relation symbols are to be translated into an appropriate semiring.

In this paper, we shall consider MSO-definable linear optimization problems, also called LinMSO-definable optimization problems. It is not hard to see that the methods in this paper extend to other classes of MSO-definable problems, such as counting and enumeration problems. See, e.g., [10, Chapter 6] for an overview of MSO-definable problems and their algorithmic applications.

Definition 2 (LinMSO-definable Optimization Problem)**.** Let τ be a relational vocabulary, $\bar{R} = \{R_1, \ldots, R_l\} \subseteq \tau$ be a set of unary relation symbols, $\varphi \in \mathrm{MSO}(\tau)$, and $\tau' = \tau \setminus \bar{R}$. Let $\alpha_1, \ldots, \alpha_l \in \mathbf{Z}$ and $\min \emptyset := \infty$.

Then we call the problem of, given a τ'-structure $\mathscr{A}$, computing

$$\min\left\{ \sum_{k=1}^{l} \alpha_k |U_k| \;\middle|\; U_i \subseteq A,\ 1 \leq i \leq l, \text{ and } (\mathscr{A}, U_1, \ldots, U_l) \models \varphi \right\}$$

a *LinMSO-definable optimization problem*.

Example 2. *Consider the following formulas:*

$$vc(R) = \forall x \forall y (\neg adj(x,y) \vee x \in R \vee y \in R) \in MSO(\tau_{Graph} \cup \{R\})$$
$$ds(R) = \forall x (x \in R \vee \exists y (y \in R \wedge adj(x,y))) \in MSO(\tau_{Graph} \cup \{R\})$$
$$\begin{aligned} 3col = \exists R_1 \exists R_2 \exists R_3 \Bigg(\forall x \Bigg(\bigvee_{i=1}^{3} (x \in R_i) \wedge \bigwedge_{i \neq j} (\neg x \in R_i \vee \neg x \in R_j) \Bigg) \wedge \\ \forall x \forall y \Bigg(\neg adj(x,y) \vee \bigwedge_{i=1}^{3} (\neg x \in R_i \vee \neg y \in R_i) \Bigg) \Bigg) \in MSO(\tau_{Graph}) \end{aligned}$$

Then, given a τ_{Graph}-structure $\mathscr{G}$,

$$\begin{aligned} &\min\{\, |C| \;\big|\; C \subseteq A \wedge (\mathscr{G}, C) \models vc \,\}, \\ &\min\{\, |D| \;\big|\; D \subseteq A \wedge (\mathscr{G}, D) \models ds \,\}, \text{ and} \\ &\min\{\, 0 \;\big|\; \mathscr{G} \models 3col \,\} \end{aligned}$$

encode the well known graph problems Minimum Vertex Cover, Minimum Dominating Set, *and* 3-Colorability, *respectively.*

2. Model Checking Games

The semantics of MSO in the classical sense (cf. [44, 35]) can be characterized using a two player pebble game, called the *Hintikka game* or *model checking game*, cf. [26, 27, 28].

A pebble *game* $\mathcal{G} = (P, M, P_0, P_1, p_0)$ between two players, say Player 0 and Player 1, consists of a finite set P of *positions*, two disjoint sets $P_0, P_1 \subseteq P$ assigning positions to the two players, an *initial position* $p_0 \in P$, and an acyclic binary relation $M \subseteq P \times P$, which specifies the valid *moves* in the game. We only allow moves from positions assigned to one of the two players, i.e., we require $p \in P_0 \cup P_1$ for all $(p, p') \in M$. On the other hand, we do allow that positions without outgoing moves are assigned to players. Let $|\mathcal{G}| := |P|$ be the *size* of $\mathcal{G}$.

For $p \in P$, we let $next_{\mathcal{G}}(p) = \{\, p' \in P \mid (p, p') \in M \,\}$ be the set of positions reachable from p via a move in M. For any position $p \in next_{\mathcal{G}}(p_0)$ we let $subgame_{\mathcal{G}}(p) = (P, M, P_0, P_1, p)$ be a *subgame* of $\mathcal{G}$, which is issued from the new initial position p. The set of all subgames of $\mathcal{G}$ is denoted by $subgames(\mathcal{G})$. If $\mathcal{G}$ is clear from the context, we usually omit the subscript and write $next(p)$ and $subgame(p)$.

A *play* of $\mathcal{G}$ is a maximal sequence $(p_0, \ldots, p_l)$ of positions $p_0, \ldots, p_{l-1} \in P_0 \cup P_1$, such that between any subsequent positions p_i and p_{i+1} there is a valid move, i.e., $(p_i, p_{i+1}) \in M$ for $0 \leq i \leq l-1$. Such a play is said to have l *rounds* and to *end* in position p_l.

The rules of the game are that in the ith round of the play, where $1 \leq i \leq l-1$, the player assigned to position p_i has to place a valid *move*, i.e., has to choose the next position $p_{i+1} \in \mathit{next}(p_i)$. If no such position p_{i+1} exists, or the position p_i is not assigned to either of the players, the play ends. If the play ends in a position p_l with $p_l \in P_i$, where $i \in \{0, 1\}$, then the other player, Player $(1-i)$, *wins* the play. If, however, the play ends in a position p_l with $p_l \notin P_0 \cup P_1$, then there is a draw and none of the players wins the play. The goal of game is to force the other player into a position where they cannot move.

We say that a player has a *winning strategy on* $\mathcal{G}$, if and only if they can win every play of the game irrespective of the choices of the other player. For instance, Player 0 has a winning strategy on $\mathcal{G}$ if and only if either

- $p_0 \in P_0$ and there is a move $(p_0, p_1) \in M$ such that Player 0 has a winning strategy on $\mathit{subgame}_{\mathcal{G}}(p_1)$; or
- $p_0 \in P_1$ and Player 0 has a winning strategy on $\mathit{subgame}_{\mathcal{G}}(p_1)$ for all moves $(p_0, p_1) \in M$. Note that this includes the case that Player 1 cannot move at all.

A game $\mathcal{G}$ is said to be *determined* or *well-founded* if either one of the players has a winning strategy on $\mathcal{G}$, otherwise $\mathcal{G}$ is *undetermined*.

We fix two special games $\bot$ and $\top$ on which the first player and the second player, respectively, have winning strategies. One can efficiently test whether one of the player has a winning strategy on a game $\mathcal{G}$, cf., [27, 28]. Algorithm 1 determines whether one of the players has a winning strategy on a game $\mathcal{G}$ and returns either $\bot$ or $\top$ if this is the case. If none of the players has a winning strategy, the algorithm returns a corresponding "proof", a list of all the plays of $\mathcal{G}$ that ended with a draw.

In the case of the model checking game, we call the two players the *falsifier* and the *verifier*. The verifier wants to prove that a formula is *true* on a structure (or, the structure *satisfies* the formula), while the falsifier tries to show that it is *false* (or, the structure does not satisfy the formula). The reader may therefore call $\top$ "true" and $\bot$ "false".

Definition 3 (Model Checking Game)**.** The (classical) *model checking game* $\mathcal{MC}(\mathscr{A}, \varphi) = (P, M, P_0, P_1, p_0)$ over a fully interpreted τ-structure $\mathscr{A}$ and a formula $\varphi \in \mathrm{MSO}(\tau)$ is defined by induction over the structure of φ as follows. Let $p_0 = (\mathscr{A}[\bar{c}^{\mathscr{A}}], \varphi)$, where $\bar{c} = \mathit{null}(\tau)$. If φ is an atomic or negated formula, then $\mathcal{MC}(\mathscr{A}, \varphi) = (\{p_0\}, \emptyset, P_0, P_1, p_0)$, where

- $p_0 \in P_0$ if and only if
  - $\varphi = R(c_1, \ldots, c_p)$ and $(c_1^{\mathscr{A}}, \ldots, c_p^{\mathscr{A}}) \in R^{\mathscr{A}}$, or
  - $\psi = \neg R(c_1, \ldots, c_p)$ and $(c_1^{\mathscr{A}}, \ldots, c_p^{\mathscr{A}}) \notin R^{\mathscr{A}}$.

Algorithm 1 Evaluating a game.

Algorithm $eval(\mathcal{G})$
Input: A game $\mathcal{G} = (P, M, P_0, P_1, p_0)$.

if $\mathcal{G} \in \{\top, \bot\}$ **then return** $\mathcal{G}$
Let $P' = \{p_0\}$, $M' = \emptyset$, $P'_0 = P_0 \cap \{p_0\}$, and $P'_1 = P_1 \cap \{p_0\}$.
for $p' \in next(p_0)$ **do**
  Let $(P'', M', P''_0, P''_1, p''_0) = eval(subgame_{\mathcal{G}}(p'))$.
  Update $P' := P' \cup P''$ and $P'_0 := P'_0 \cup P''_0$, $P'_1 := P'_1 \cup P''_1$.
  Update $M' := M' \cup M'' \cup \{(p'_0, p''_0)\}$.
Let $\mathcal{G}' = (P', M', P'_0, P'_1, p_0)$ and compute $subgames(\mathcal{G}')$.
if $p_0 \in P'_0$ **then**
  if $subgames(\mathcal{G}') = \{\top\}$ **or** $subgames(\mathcal{G}') = \emptyset$ **then return** $\top$
  if $\bot \in subgames(\mathcal{G}')$ **then return** $\bot$
if $p_0 \in P'_1$ **then**
  if $subgames(\mathcal{G}') = \{\bot\}$ **or** $subgames(\mathcal{G}') = \emptyset$ **then return** $\bot$
  if $\top \in subgames(\mathcal{G}')$ **then return** $\top$
return $\mathcal{G}'$

- $p_0 \in P_1$ if and only if
  - $\varphi = R(c_1, \ldots, c_p)$ and $(c_1^{\mathscr{A}}, \ldots, c_p^{\mathscr{A}}) \notin R^{\mathscr{A}}$, or
  - $\psi = \neg R(c_1, \ldots, c_p)$ and $(c_1^{\mathscr{A}}, \ldots, c_p^{\mathscr{A}}) \in R^{\mathscr{A}}$.

If $\varphi \in \{\forall R\psi, \exists R\psi\}$ for some relation symbol R, let $\mathscr{A}_U = (\mathscr{A}, U)$ for $U \subseteq A$ be the (τ, R)-expansion of $\mathscr{A}$ with $R^{\mathscr{A}_U} = U$, and let $\mathcal{MC}(\mathscr{A}_U, \psi) = (P_U, M_U, P_{0,U}, P_{1,U}, p_U)$ be the corresponding model checking game over $\mathscr{A}_U$ and ψ. Then $\mathcal{MC}(\mathscr{A}, \varphi) = (P, M, P_0, P_1, p_0)$, where

- $P = \{p_0\} \cup \bigcup_{U \subseteq A} P_U$,
- $M = \bigcup_{U \subseteq A} (M_U \cup \{(p_0, p_U)\})$,
- $P_0 = P'_0 \cup \bigcup_{U \subseteq A} P_{0,U}$, where $P'_0 = \{p_0\}$ iff $\varphi = \forall R\psi$ and $P'_0 = \emptyset$ otherwise,
- $P_1 = P'_1 \cup \bigcup_{U \subseteq A} P_{1,U}$, where $P'_1 = \{p_1\}$ iff $\varphi = \exists R\psi$ and $P'_1 = \emptyset$ otherwise.

If $\varphi \in \{\forall c\psi, \exists c\psi\}$ for some nullary symbol c, let $\mathscr{A}_a = (\mathscr{A}, a)$ be the (τ, c)-expansion of $\mathscr{A}$ with $c^{\mathscr{A}_a} = a \in A$, and let $\mathcal{MC}(\mathscr{A}_a, \psi) = (P_a, M_a, P_{0,a}, P_{1,a}, p_a)$ be the corresponding model checking game over $\mathscr{A}_a$ and ψ. Then $\mathcal{MC}(\mathscr{A}, \varphi) = (P, M, P_0, P_1, p_0)$, where

- $P = \{p_0\} \cup \bigcup_{a \in A} P_a$,
- $M = \bigcup_{a \in A} (M_a \cup \{(p_0, p_a)\})$,
- $P_0 = P'_0 \cup \bigcup_{a \in A} P_{0,a}$, where $P'_0 = \{p_0\}$ iff $\varphi = \forall c\psi$ and $P'_0 = \emptyset$ otherwise,
- $P_1 = P'_1 \cup \bigcup_{a \in A} P_{1,a}$, where $P'_1 = \{p_1\}$ iff $\varphi = \exists c\psi$ and $P'_1 = \emptyset$ otherwise.

If $\varphi \in \{\psi_1 \wedge \psi_2, \psi_1 \vee \psi_2\}$, let $\mathcal{MC}(\mathscr{A}, \psi) = (P_\psi, M_\psi, P_{0,\psi}, P_{1,\psi}, p_\psi)$ be the model checking game over $\mathscr{A}$ and $\psi \in \{\psi_1, \psi_2\}$. Then $\mathcal{MC}(\mathscr{A}, \varphi) = (P, M, P_0, P_1, p_0)$, where

- $P = \{p_0\} \cup \bigcup_{\psi \in \{\psi_1, \psi_2\}} P_\psi$,
- $M = \bigcup_{\psi \in \{\psi_1, \psi_2\}} (M_\psi \cup \{(p_0, p_\psi)\})$,
- $P_0 = P_0' \cup \bigcup_{\psi \in \{\psi_1, \psi_2\}} P_{0,\psi}$, where $P_0' = \{p_0\}$ iff $\varphi = \psi_1 \wedge \psi_2$ and $P_0' = \emptyset$ otherwise,
- $P_1 = P_1' \cup \bigcup_{\psi \in \{\psi_1, \psi_2\}} P_{1,\psi}$, where $P_1' = \{p_1\}$ iff $\varphi = \psi_1 \vee \psi_2$ and $P_1' = \emptyset$ otherwise.

Note that the falsifier is the *universal* player and moves on universal formulas, while the verifier is the *existential* player and moves on existential formulas. Furthermore, if the structure $\mathscr{A}$ is empty, then, by definition, $\mathscr{A} \models \forall c\psi$ and $\mathscr{A} \not\models \exists c\psi$ for all ψ. In the model checking game, this corresponds to the case that there are no moves from the current position. Consequently, the play ends and the player assigned to this position looses. On non-empty structures, each play ends in an atomic or negated atomic formula. The goal of the verifier is to make the play end in a position $(\mathscr{A}', \psi)$ with $\mathscr{A}' \models \psi$, and conversely the goal of the falsifier is to force the play into an ending position $(\mathscr{A}', \psi)$ with $\mathscr{A}' \not\models \psi$. It is well-known that the classical model checking game is well-founded [26] and that the verifier has a winning strategy on $\mathcal{MC}(\mathscr{A}, \varphi)$ if and only if $\mathscr{A} \models \varphi$, see, e.g., [27].

2.1. An Extension of the Classical Model Checking Game

We shall now consider an extension of the model checking game that has the following two central properties:

- It is defined for partially interpreted structures; and
- it is "well-defined" under taking the union of structures in the sense that if one of the players has a winning strategy on the game on $\mathscr{A}$ and φ, then the same player has a winning strategy in the game on $\mathscr{A} \cup \mathscr{B}$ and φ for all structures $\mathscr{B}$ compatible with $\mathscr{A}$.

Before we give the formal definition of the new game, let us briefly mention why we require these properties: Recall that we want to use the model checking game $\mathcal{MC}(\mathscr{A}, \varphi)$ to decide algorithmically whether a τ-structure $\mathscr{A}$ holds on a formula $\varphi \in \mathrm{MSO}(\tau)$. If φ contains set quantifiers, then there is a number of positions in $\mathcal{MC}(\mathscr{A}, \varphi)$ that grows exponentially with the size of A. In order to avoid exponential running time on structures of bounded treewidth, a tree decomposition $(\mathcal{T}, \mathcal{X})$ of $\mathscr{A}$, where $\mathcal{T} = (T, F)$, is traversed bottom-up by a dynamic programming algorithm. At a node $i \in T$, we only consider the substructure $\mathscr{A}_i$ of $\mathscr{A}$. Let $\mathscr{A}'$ be some expansion of $\mathscr{A}$. Then $\mathscr{A}[A_i]$ is in general not fully interpreted, which explains the first requirement.

For the second requirement, note that for each $i \in T$ there is a τ-structure $\mathscr{B}_i$, such that $\mathscr{A}$ can be written as $\mathscr{A} = \mathscr{A}_i \cup \mathscr{B}_i$. The structure $\mathscr{B}_i$ is sometimes called the "future" of $\mathscr{A}_i$ in the literature. Therefore, if one of the players has a winning strategy in the game on $\mathscr{A}_i$ and φ, we require that the same player has a winning strategy on $\mathscr{A} = \mathscr{A}_i \cup \mathscr{B}_i$ and φ.

In order to make the inductive construction work, we additionally need to distinguish the nodes in the "current" bag X_i of the tree decomposition. The game therefore additionally depends on a given set $X = X_i \subseteq A$.

Definition 4 (Extended Model Checking Game)**.** The *extended model checking game* $\mathcal{EMC}(\mathscr{A}, X, \varphi) = (P, M, P_0, P_1, p_0)$ over a τ-structure $\mathscr{A}$, a set $X \subseteq A$, and a formula $\varphi \in \mathrm{MSO}(\tau)$ is defined by induction over the structure of φ as follows. Let $p_0 = (\mathscr{A}[X \cup \bar{c}^{\mathscr{A}}], X, \varphi)$, where $\bar{c} = \mathit{null}(\tau)$. If φ is an atomic or negated formula, then $\mathcal{EMC}(\mathscr{A}, \varphi) = (\{p_0\}, \emptyset, P_0, P_1, p_0)$, where

- $p_0 \in P_0$ if and only if either
  - $\varphi = R(c_1, \ldots, c_p)$, such that $\{c_1, \ldots, c_p\} \subseteq \mathit{interpreted}(\mathscr{A})$, and $(c_1^{\mathscr{A}}, \ldots, c_p^{\mathscr{A}}) \in R^{\mathscr{A}}$, or
  - $\varphi = \neg R(c_1, \ldots, c_p)$, such that $\{c_1, \ldots, c_p\} \subseteq \mathit{interpreted}(\mathscr{A})$, and $(c_1^{\mathscr{A}}, \ldots, c_p^{\mathscr{A}}) \notin R^{\mathscr{A}}$.
- $p_0 \in P_1$ if and only if either
  - $\varphi = R(c_1, \ldots, c_p)$, such that $\{c_1, \ldots, c_p\} \subseteq \mathit{interpreted}(\mathscr{A})$, and $(c_1^{\mathscr{A}}, \ldots, c_p^{\mathscr{A}}) \notin R^{\mathscr{A}}$, or
  - $\varphi = \neg R(c_1, \ldots, c_p)$, such that $\{c_1, \ldots, c_p\} \subseteq \mathit{interpreted}(\mathscr{A})$, and $(c_1^{\mathscr{A}}, \ldots, c_p^{\mathscr{A}}) \in R^{\mathscr{A}}$.

If $\varphi \in \{\forall R\psi, \exists R\psi\}$ for some relation symbol R, or $\varphi \in \{\psi_1 \wedge \psi_2, \psi_1 \vee \psi_2\}$, then $\mathcal{EMC}(\mathscr{A}, X, \varphi)$ is defined analogously to $\mathcal{MC}(\mathscr{A}, \varphi)$.

If $\varphi \in \{\forall c\psi, \exists c\psi\}$ for some nullary symbol c, let $\mathscr{A}_u = (\mathscr{A}, u)$ be the (τ, c)-expansion of $\mathscr{A}$ with $c^{\mathscr{A}_u} = u \in A \cup \{\mathrm{nil}\}$, and let $\mathcal{EMC}(\mathscr{A}_u, X, \psi) = (P_u, M_u, P_{0,u}, P_{1,u}, p_u)$ be the corresponding extended model checking game over $\mathscr{A}_u$ and ψ. Then $\mathcal{EMC}(\mathscr{A}, X, \varphi) = (P, M, P_0, P_1, p_0)$, where

- $P = \{p_0\} \cup \bigcup_{u \in A \cup \{\mathrm{nil}\}} P_u$,
- $M = \bigcup_{u \in A \cup \{\mathrm{nil}\}} (M_u \cup \{(p_0, p_u)\})$,
- $P_0 = P_0' \cup \bigcup_{u \in A \cup \{\mathrm{nil}\}} P_{0,u}$, where $P_0' = \{p_0\}$ iff $\varphi = \forall c\psi$ and $P_0' = \emptyset$ otherwise,
- $P_1 = P_1' \cup \bigcup_{u \in A \cup \{\mathrm{nil}\}} P_{1,u}$, where $P_1' = \{p_1\}$ iff $\varphi = \exists c\psi$ and $P_1' = \emptyset$ otherwise.

For the games we consider throughout this paper, one can derive from a position $p \in P$ whether $p \in P_0$ or $p \in P_1$ (cf., the definitions of $\mathcal{MC}$ and $\mathcal{EMC}$).

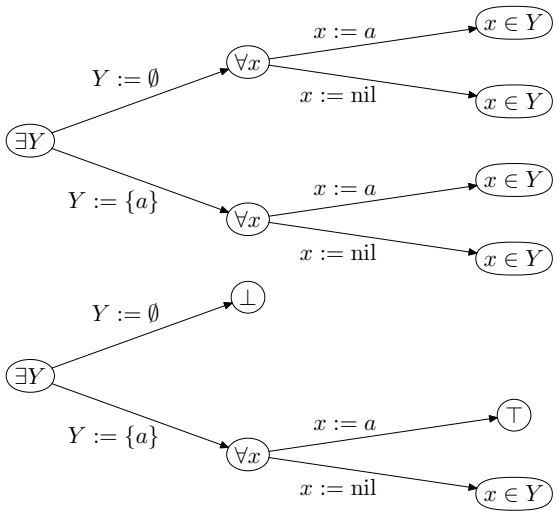

Figure 1: Top: simplified schematic of $\mathcal{EMC}(\mathscr{A}, \emptyset, \varphi)$ for the structure $\mathscr{A}$ with universe $A = \{a\}$ and $\varphi = \exists Y \forall x (x \in Y)$. Bottom: $\mathit{eval}(\mathcal{EMC}(\mathscr{A}, \emptyset, \varphi))$. The lower branch witnesses a play that ends with a draw.

To avoid cluttered notation, we shall therefore usually omit the sets P_0 and P_1 from the tuple (P, M, P_0, P_1, p_0) and identify games with the triple (P, M, p_0). Figure 1 shows a simplified schematic of an extended model checking game and the result after an application of the evaluation algorithm *eval*.

If $\mathscr{A}$ is a fully interpreted structure, $\mathcal{MC}(\mathscr{A}, \varphi)$ can be embedded into $\mathcal{EMC}(\mathscr{A}, X, \varphi)$ such that for each play of $\mathcal{MC}(\mathscr{A}, \varphi)$ there is a corresponding, equivalent play of $\mathcal{EMC}(\mathscr{A}, X, \varphi)$. Algorithm 2 effectively computes this embedding (Lemma 1). Furthermore, if $\mathcal{EMC}(\mathscr{A}, X, \varphi)$ is determined, then so is $\mathcal{MC}(\mathscr{A}, \varphi)$ (Lemma 3).

Lemma 1. *Let $\mathscr{A}$ be a fully interpreted τ-structure, $X \subseteq A$, and $\varphi \in MSO(\tau)$. Then, using Algorithm 2, we have*

$$\mathcal{MC}(\mathscr{A}, \varphi) = \mathit{convert}(\mathcal{EMC}(\mathscr{A}, X, \varphi)).$$

Proof. The proof is an induction over the structure of φ. For atomic or negated atomic formulas, the statement trivially holds by definition of $\mathcal{MC}(\mathscr{A}, \varphi)$, since $\mathit{subgames}(\mathcal{EMC}(\mathscr{A}, X, \varphi)) = \emptyset$. Let $\mathcal{G} = \mathcal{EMC}(\mathscr{A}, X, \varphi) = (P, M, p_0)$.

Let $\varphi \in \{\forall R\psi, \exists R\psi\}$ or $\varphi \in \{\psi_1 \wedge \psi_2, \psi_1 \vee \psi_2\}$ and $\psi \in \{\psi_1, \psi_2\}$ and consider $p = (\mathscr{H}, X, \psi) \in \mathit{next}_{\mathcal{G}}(p_0)$. We have $\mathit{subgame}_{\mathcal{G}}(p) = \mathcal{EMC}(\mathscr{A}', X, \psi)$, where either $\mathscr{A}' = (\mathscr{A}, U)$ is an (τ, R)-expansion of $\mathscr{A}$ for some $U \subseteq A$, or $\mathscr{A}' = \mathscr{A}$, respectively. Since $\mathscr{A}$ is fully interpreted, $\mathscr{A}'$ is fully interpreted, and we obtain $\mathcal{MC}(\mathscr{A}', \psi) = \mathit{convert}(\mathcal{EMC}(\mathscr{A}', X, \psi))$ by the induction hypothesis.

If otherwise $\varphi \in \{\forall c\psi, \exists c\psi\}$, consider $p = (\mathscr{H}, X, \psi) \in \mathit{next}_{\mathcal{G}}(p_0)$. By definition, $\mathit{subgame}_{\mathcal{G}}(p) = \mathcal{EMC}(\mathscr{A}', X, \psi)$, where $\mathscr{A}'$ is a (τ, c)-expansion of $\mathscr{A}$

Algorithm 2 Converting $\mathcal{EMC}$ to $\mathcal{MC}$

Algorithm *convert*$(\mathcal{G})$
Input: A game $\mathcal{G} = (P, M, p_0)$.

if $\mathcal{G} \in \{\top, \bot\}$ **then return** $\mathcal{G}$.
Let $p_0 = (\mathscr{H}, X, \varphi)$ and $\bar{c} = \mathit{null}(\mathit{vocabulary}(\mathscr{H}))$.
Let $p_0' = (\mathscr{H}[\bar{c}^{\mathscr{H}}], \varphi)$, $P' = \{p_0'\}$, and $M' = \emptyset$.
for $p_1 = (\mathscr{H}_1, X, \psi) \in \mathit{next}(p_0)$ s.t. $\mathscr{H}_1$ is fully interpreted **do**
  Let $(P_1', M_1', p_1') = \mathit{convert}(\mathit{subgame}_{\mathcal{G}}(p_1))$.
  Update $P' := P' \cup P_1'$ and $M' := M' \cup M_1' \cup \{(p_0', p_1')\}$.
return (P', M', p_0')

with $c^{\mathscr{A}'} \in A \cup \{\text{nil}\}$. If all constant symbols are interpreted in $\mathscr{H}$, then $c^{\mathscr{A}'} \neq \text{nil}$, i.e., $\mathscr{A}'$ is fully interpreted. By the induction hypothesis we get $\mathcal{MC}(\mathscr{A}', \psi) = \mathit{convert}(\mathcal{EMC}(\mathscr{A}', X, \psi))$.

Together, the statement follows. □

We now prove that if an extended model game is determined, then the corresponding player can win the game without using any further "nil-moves". This will be useful in the proof of Lemma 3.

Lemma 2. *Let $\mathscr{A}_1$ and $\mathscr{A}_2$ be τ-structures with $A_1 = A_2$ and $c \in \mathit{null}(\tau)$, such that $c^{\mathscr{A}_1} = \text{nil}$, $R^{\mathscr{A}_1} = R^{\mathscr{A}_2}$ for all $R \in \mathit{rel}(\tau)$ and $d^{\mathscr{A}_1} = d^{\mathscr{A}_2}$ for all $d \in \mathit{null}(\tau) \setminus \{c\}$. Let $\varphi \in \mathit{MSO}(\tau)$.*

If $\mathit{eval}(\mathcal{EMC}(\mathscr{A}_1, X, \varphi)) \in \{\top, \bot\}$, then $A_1 \neq \emptyset$ and

$$\mathit{eval}(\mathcal{EMC}(\mathscr{A}_1, X, \varphi)) = \mathit{eval}(\mathcal{EMC}(\mathscr{A}_2, X, \varphi)).$$

Before we give the formal proof, consider the following high-level argument: Suppose that $\mathit{eval}(\mathcal{EMC}(\mathscr{A}_1, X, \varphi)) = \top$. Then there is at least one play of the game $\mathcal{EMC}(\mathscr{A}_1, X, \varphi)$ that is won by the verifier. Consider an arbitrary play $(p_0, \ldots, p_l)$ won by the verifier and let $p_l = (\mathscr{H}, X, \psi)$. Since p_l is assigned to the falsifier, all constant symbols occurring in ψ are interpreted and hence different from c. The verifier can therefore win the game without depending on formulas where c occurs.

Proof. The proof is an induction over the structure of φ.

Let $\mathit{eval}(\mathcal{EMC}(\mathscr{A}_1, X, \varphi)) \in \{\top, \bot\}$. If φ is an atomic or negated formula, say $\varphi = R(c_1, \ldots, c_p)$, then $\{c_1, \ldots, c_p\} \subseteq \mathit{interpreted}(\mathscr{A}_1)$. Therefore, $A_1 \neq \emptyset$ and for all $1 \leq i \leq p$, we have $c \neq c_i$ and $c_i^{\mathscr{A}_1} = c_i^{\mathscr{A}_2}$, which implies $\mathit{eval}(\mathcal{EMC}(\mathscr{A}_1, X, \varphi)) = \mathit{eval}(\mathcal{EMC}(\mathscr{A}_2, X, \varphi))$.

If $\varphi \in \{\forall R\psi, \exists R\psi\}$ for a relation symbol R, let $U \subseteq A$ and $\mathscr{A}_1'$, $\mathscr{A}_2'$ be the (τ, R)-expansions of $\mathscr{A}_1$ and $\mathscr{A}_2$, respectively, with $R^{\mathscr{A}_1'} = R^{\mathscr{A}_2'} = U$. Then by the induction hypothesis $\mathit{eval}(\mathcal{EMC}(\mathscr{A}_1', X, \psi)) = \mathit{eval}(\mathcal{EMC}(\mathscr{A}_2', X, \psi))$ if $\mathit{eval}(\mathcal{EMC}(\mathscr{A}_1', X, \psi)) \in \{\top, \bot\}$.

Similarly, if $\varphi \in \{\forall d\psi, \exists d\psi\}$ for a nullary symbol d, let $\mathscr{A}_1'$ and $\mathscr{A}_2'$ be (τ, d)-expansions of $\mathscr{A}_1$ and $\mathscr{A}_2$, respectively, such that $d^{\mathscr{A}_1'} = d^{\mathscr{A}_2'}$. Then

by the induction hypothesis $\mathit{eval}(\mathcal{EMC}(\mathscr{A}_1', X, \psi)) = \mathit{eval}(\mathcal{EMC}(\mathscr{A}_2', X, \psi))$, if $\mathit{eval}(\mathcal{EMC}(\mathscr{A}_1', X, \psi)) \in \{\top, \bot\}$.

Finally, if $\varphi \in \{\psi_1 \wedge \psi_2, \psi_1 \vee \psi_2\}$, then from $\mathit{eval}(\mathcal{EMC}(\mathscr{A}_1, X, \psi)) \in \{\top, \bot\}$, where $\psi \in \{\psi_1, \psi_2\}$, we get $\mathit{eval}(\mathcal{EMC}(\mathscr{A}_1', X, \psi)) = \mathit{eval}(\mathcal{EMC}(\mathscr{A}_2', X, \psi))$.

Together, the statement of the lemma follows. □

We can now prove that if some player has a winning strategy in the extended model checking game, then the same player has a winning strategy in the classical model checking game.

Lemma 3. *Let $\mathscr{A}$ be a fully interpreted τ-structure, $X \subseteq A$, and $\varphi \in \mathit{MSO}(\tau)$. If $\mathit{eval}(\mathcal{EMC}(\mathscr{A}, X, \varphi)) \in \{\top, \bot\}$, then*

$$\mathit{eval}(\mathcal{MC}(\mathscr{A}, \varphi)) = \mathit{eval}(\mathcal{EMC}(\mathscr{A}, X, \varphi)).$$

Proof. The proof is an induction over the structure of φ.

Suppose $\mathit{eval}(\mathcal{EMC}(\mathscr{A}, X, \varphi)) = \top$ (the case $\bot$ is shown analogously). If φ is an atomic or negated atomic formula, then the statement clearly holds. If $\varphi = \psi_1 \wedge \psi_2$, then for each $\psi \in \{\psi_1, \psi_2\}$ we have $\mathit{eval}(\mathcal{EMC}(\mathscr{A}, X, \psi)) = \top$. This implies $\mathit{eval}(\mathcal{MC}(\mathscr{A}, \psi)) = \top$ by the induction hypothesis, and therefore $\mathit{eval}(\mathcal{MC}(\mathscr{A}, \varphi)) = \top$.

Similarly, if $\varphi = \forall R\psi$ for a relation symbol R, then $\mathit{eval}(\mathcal{EMC}(\mathscr{A}', X, \psi)) = \top$ for each (τ, R)-expansion $\mathscr{A}'$ of $\mathscr{A}$, each of which is fully interpreted. We get $\mathit{eval}(\mathcal{MC}(\mathscr{A}', \psi)) = \top$ by the induction hypothesis. Hence, $\mathit{eval}(\mathcal{MC}(\mathscr{A}, \varphi)) = \top$.

If $\varphi = \forall c\psi$ for a nullary symbol c, then $\mathit{eval}(\mathcal{EMC}(\mathscr{A}', X, \psi)) = \top$ for each fully interpreted (τ, c)-expansion $\mathscr{A}'$ of $\mathscr{A}$. This implies $\mathit{eval}(\mathcal{MC}(\mathscr{A}', \psi)) = \top$ by the induction hypothesis, and therefore $\mathit{eval}(\mathcal{MC}(\mathscr{A}, \varphi)) = \top$.

If $\varphi = \psi_1 \vee \psi_2$, then there is $\psi \in \{\psi_1, \psi_2\}$ with $\mathit{eval}(\mathcal{EMC}(\mathscr{A}, X, \psi)) = \top$. We get $\mathit{eval}(\mathcal{MC}(\mathscr{A}, \psi)) = \top$ by the induction hypothesis, and therefore $\mathit{eval}(\mathcal{MC}(\mathscr{A}, \varphi)) = \top$.

Similarly, if $\varphi = \exists R\psi$ for a relation symbol R, then there is a (τ, R)-expansion $\mathscr{A}'$ of $\mathscr{A}$ with $\mathit{eval}(\mathcal{EMC}(\mathscr{A}', X, \psi)) = \top$. Since $\mathscr{A}$ is fully interpreted, $\mathscr{A}'$ is fully interpreted. Using the induction hypothesis, we have $\mathit{eval}(\mathcal{MC}(\mathscr{A}', \psi)) = \top$ and therefore $\mathit{eval}(\mathcal{MC}(\mathscr{A}, \varphi)) = \top$.

Finally, if $\varphi = \exists c\psi$ for a nullary symbol c, then there is a (τ, c)-expansion $\mathscr{A}'$ of $\mathscr{A}$ with $\mathit{eval}(\mathcal{EMC}(\mathscr{A}', X, \psi)) = \top$. By Lemma 2, we can assume $c^{\mathscr{A}} \neq \mathrm{nil}$. Then $\mathscr{A}'$ is fully interpreted and we get $\mathit{eval}(\mathcal{MC}(\mathscr{A}', \psi)) = \top$ by the induction hypothesis. Therefore $\mathit{eval}(\mathcal{MC}(\mathscr{A}, \varphi)) = \top$. □

We can significantly strengthen this statement further: If $\mathcal{EMC}(\mathscr{A}, X, \varphi)$ is determined, then $\mathcal{EMC}(\mathscr{A} \cup \mathscr{B}, X, \varphi)$ is also determined for *all* $\mathscr{B}$ compatible with $\mathscr{A}$. Note that the union $\mathscr{A} \cup \mathscr{B}$ arises on *join* or *introduce* nodes i of the tree decomposition, where $X = X_i$ is the current bag, cf., Figure 2.

Recall, for instance, the example 3-Colorability from the introduction: If a subgraph $\mathscr{A}'$ of a graph $\mathscr{A}$ is not three-colorable, then clearly $\mathscr{A}$ is not three-colorable either. The following lemma formalizes this observation.

Let us give a brief high-level explanation before we state the lemma and give its proof. Roughly speaking, if $\mathcal{G} = \mathcal{EMC}(\mathscr{A}, X, \varphi)$ is determined, then moves to objects $b \in B \setminus A$ in $\mathcal{G}' = \mathcal{EMC}(\mathscr{A} \cup \mathscr{B}, X, \varphi)$ are either "irrelevant" for a player's strategy or already "sufficiently" captured by moves to nil (cf., Lemma 2). If therefore one of the players, say the falsifier, has a winning strategy in $\mathcal{G}$, then in some sense this winning strategy carries over to $\mathcal{G}'$. In the case of 3-Colorability, if $\mathscr{A}$ is not three-colorable, then the falsifier has a winning strategy on $\mathcal{EMC}(\mathscr{A}, X, \mathit{3col})$: No matter which three sets the verifier chooses, either these sets are not a partition or not independent sets. In either case there are witnessing vertices that the falsifier can choose. Thus, no matter which subsets the verifier chooses in $\mathcal{G}' = \mathcal{EMC}(\mathscr{A} \cup \mathscr{B}, X, \mathit{3col})$, the falsifier can then choose the same witnessing vertices to win each play of $\mathcal{G}'$.

Lemma 4 (Introduce)**.** *Let $\mathscr{A}$ and $\mathscr{B}$ be compatible τ-structures with $B = A \uplus \{b\}$. Let $X \subseteq A$ and $\varphi \in MSO(\tau)$. Let $\mathcal{G} = \mathcal{EMC}(\mathscr{A}, X, \varphi)$ and $\mathcal{G}' = \mathcal{EMC}(\mathscr{B}, X \cup \{b\}, \varphi)$.*

1. *If $\mathit{eval}(\mathcal{G}) = \top$, then $\mathit{eval}(\mathcal{G}') = \top$.*
2. *If $\mathit{eval}(\mathcal{G}) = \bot$, then $\mathit{eval}(\mathcal{G}') = \bot$.*

Proof. We prove the lemma by induction over the structure of φ. Let $\bar{c} = \mathit{null}(\tau)$. Let $\mathcal{G} = (P, M, p_0)$ and $\mathcal{G}' = (P', M', p_0')$ with $p_0 = (\mathscr{H}, X, \varphi)$ and $p_0' = (\mathscr{H}', X \cup \{b\}, \varphi)$, where $\mathscr{H} = \mathscr{A}[X \cup \bar{c}^{\mathscr{A}}]$ and $\mathscr{H}' = \mathscr{B}[X \cup \{b\} \cup \bar{c}^{\mathscr{B}}]$. Suppose $\mathit{eval}(\mathcal{G}) = \top$ (the second case $\mathit{eval}(\mathcal{G}) = \bot$ is proven analogously).

Let $\varphi = R(c_1, \ldots, c_p)$ or $\varphi = \neg R(c_1, \ldots, c_p)$ for a relation symbol $R \in \tau$. We have $\mathit{eval}(\mathcal{G}) = \top$, and hence, by definition $c_i \in \mathit{interpreted}(\mathscr{A})$ for all $1 \leq i \leq p$. Here, $c_i^{\mathscr{H}} = c_i^{\mathscr{A}} = c_i^{\mathscr{B}} = c_i^{\mathscr{H}'}$ for all $1 \leq i \leq p$, since $\mathscr{A}$ and $\mathscr{B}$ are compatible, and therefore $R^{\mathscr{H}} = R^{\mathscr{H}'} \cap H^p$, since $H = H' \setminus \{b\}$. Hence, $(c_1^{\mathscr{H}}, \ldots, c_p^{\mathscr{H}}) \in R^{\mathscr{H}}$ if and only if $(c_1^{\mathscr{H}'}, \ldots, c_p^{\mathscr{H}'}) \in R^{\mathscr{H}'}$, and thus $\mathit{eval}(\mathcal{G}') = \top$.

Assume now that $\varphi = \psi_1 \wedge \psi_2$ or $\varphi = \psi_1 \vee \psi_2$. By definition, for each $\psi \in \{\psi_1, \psi_2\}$ there is a subgame $\mathcal{G}_\psi = \mathcal{EMC}(\mathscr{A}, X, \psi) \in \mathit{subgames}(\mathcal{G})$ and a subgame $\mathcal{G}'_\psi = \mathcal{EMC}(\mathscr{B}, X \cup \{b\}, \psi) \in \mathit{subgames}(\mathcal{G}')$. By the induction hypothesis, $\mathit{eval}(\mathcal{G}'_\psi) = \top$ if $\mathit{eval}(\mathcal{G}_\psi) = \top$, and hence $\mathit{eval}(\mathcal{G}') = \top$ if $\mathit{eval}(\mathcal{G}) = \top$.

If $\varphi = \forall R\psi$ or $\varphi = \exists R\psi$, then for each $U \subseteq A$ there is a subgame $\mathcal{G}_U = \mathcal{EMC}((\mathscr{A}, U), X, \psi)) \in \mathit{subgames}(\mathcal{G})$, and for each $U' \subseteq B$ there is a subgame $\mathcal{G}'_{U'} = \mathcal{EMC}((\mathscr{B}, U'), X \cup \{b\}, \psi)) \in \mathit{subgames}(\mathcal{G}')$.

If $\varphi = \forall R\psi$, consider an arbitrary $U' \subseteq B$ and let $U = U' \setminus \{b\}$. We know, by definition of $\mathit{eval}(\mathcal{G})$, that $\mathit{eval}(\mathcal{G}_U) = \top$. Furthermore, $(\mathscr{A}, U)$ and $(\mathscr{B}, U')$ are compatible, and therefore, by the induction hypothesis, also $\mathit{eval}(\mathcal{G}'_{U'}) = \top$. Therefore, $\mathit{eval}(\mathcal{G}'_{U'}) = \top$ for all $U' \subseteq B$, and hence $\mathit{eval}(\mathcal{G}') = \top$.

If otherwise $\varphi = \exists R\psi$, then there is some $U \subseteq A$ such that $\mathit{eval}(\mathcal{G}_U) = \top$. Since $(\mathscr{A}, U)$ and $(\mathscr{B}, U)$ are compatible, $\mathit{eval}(\mathcal{G}'_U) = \top$ by the induction hypothesis. Therefore, $\mathit{eval}(\mathcal{G}') = \top$.

If $\varphi = \forall c\psi$, consider an arbitrary (τ, c)-expansion $\mathscr{B}'$ of $\mathscr{B}$ and let $\mathscr{A}' := \mathscr{B}'[A]$. Note that if $c^{\mathscr{B}'} \neq b$, then $c^{\mathscr{A}'} = c^{\mathscr{B}'} \in A$, and if $c^{\mathscr{B}'} = b$ or $c^{\mathscr{B}'} = \text{nil}$, then $c^{\mathscr{A}'} = \text{nil}$. In either case, $\mathscr{A}'$ and $\mathscr{B}'$ are compatible. We know, by

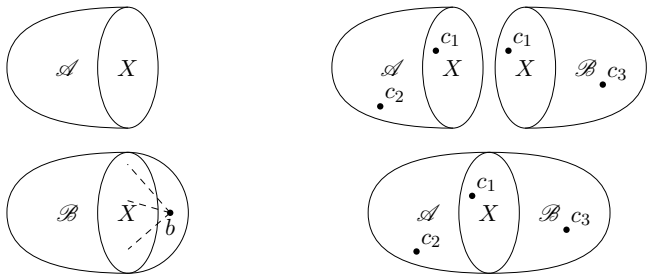

Figure 2: Introduce (left): If $\mathscr{A}$ and $\mathscr{B}$ are such that $\mathscr{A} = \mathscr{B}[A]$, then winning strategies for $\mathcal{EMC}(\mathscr{A}, X, \varphi)$ carry over to $\mathcal{EMC}(\mathscr{B}, X \cup \{b\}, \varphi)$. Join/union (right): If $\mathscr{A}$ and $\mathscr{B}$ are compatible, then winning strategies for $\mathcal{EMC}(\mathscr{A}, X, \varphi)$ carry over to $\mathcal{EMC}(\mathscr{A} \cup \mathscr{B}, X, \varphi)$.

definition of $eval(\mathcal{G})$, that $eval(\mathcal{EMC}(\mathscr{A}', X, \psi)) = \top$. Hence, by the induction hypothesis, $eval(\mathcal{EMC}(\mathscr{B}', X \cup \{b\}, \psi)) = \top$. All in all, $eval(\mathcal{G}') = \top$.

Assume now that $\varphi = \exists c\psi$. Since $eval(\mathcal{G}) = \top$, we know that there is a (τ, c)-expansion $\mathscr{A}'$ of $\mathscr{A}$, such that $eval(\mathcal{EMC}(\mathscr{A}', X, \psi)) = \top$. Let $\mathscr{B}'$ the (τ, c)-expansion of $\mathscr{B}$ with $c^{\mathscr{B}'} = c^{\mathscr{A}'}$. Then $\mathscr{A}'$ and $\mathscr{B}'$ are compatible, and using the induction hypothesis as above, we obtain $eval(\mathcal{G}') = \top$. □

Corollary 1. *Let $\mathscr{A}$ and $\mathscr{B}$ be compatible τ-structures with $A \subseteq B$. Let $X \subseteq A$ and $\varphi \in MSO(\tau)$. Let $\mathcal{G} = \mathcal{EMC}(\mathscr{A}, X, \varphi)$ and $\mathcal{G}' = \mathcal{EMC}(\mathscr{B}, X \cup (B \setminus A), \varphi)$.*

1. *If $eval(\mathcal{G}) = \top$, then $eval(\mathcal{G}') = \top$.*
2. *If $eval(\mathcal{G}) = \bot$, then $eval(\mathcal{G}') = \bot$.*

Proof. We use Lemma 4 and induction over $|B \setminus A|$. Let $eval(\mathcal{G}) \in \{\top, \bot\}$.

If $B \setminus A = \emptyset$ and therefore $\mathscr{A} = \mathscr{B}$, the statement clearly holds. Otherwise, consider $b \in B \setminus A$ and let $\mathscr{A}' = (\mathscr{A} \cup \mathscr{B})[A \cup \{b\}]$. From $eval(\mathcal{G}) \in \{\top, \bot\}$ we get $eval(\mathcal{EMC}(\mathscr{A}', X \cup \{b\}, \varphi) = eval(\mathcal{G})$ by Lemma 4.

We can now use the induction hypothesis on $\mathscr{A}'$, $\mathscr{B}$ and $X \cup \{b\}$, since $\mathscr{A}'$ and $\mathscr{B}$ a compatible and $|B \setminus A'| < |B \setminus A|$, and obtain $eval(\mathcal{G}') = eval(\mathcal{G})$. □

The *forget* operation at a node i of a tree decomposition does not change the underlying structure $\mathscr{A}_i$. It is therefore not surprising that any winning strategies carry over.

Lemma 5 (Forget)**.** *Let $\mathscr{A}$ be a τ-structure, $X' \subseteq X \subseteq A$ and $\varphi \in MSO(\tau)$. Let $\mathcal{G} = \mathcal{EMC}(\mathscr{A}, X, \varphi)$ and $\mathcal{G}' = \mathcal{EMC}(\mathscr{A}, X', \varphi)$.*

1. *If $eval(\mathcal{G}) = \top$, then $eval(\mathcal{G}') = \top$.*
2. *If $eval(\mathcal{G}) = \bot$, then $eval(\mathcal{G}') = \bot$.*

Proof. Let $\mathcal{G} = (P, M, P_0, P_1, p_0)$ and $\mathcal{G}' = (P', M', P_0', P_1', p_0')$. It is not hard to see that $\mathcal{G}$ and $\mathcal{G}'$ are almost identical, the only difference being slightly differently labeled positions: By definition, $p_0 = (\mathscr{H}, X, \varphi)$ and $p_0' = (\mathscr{H}', X', \varphi)$, where $\mathscr{H} = \mathscr{A}[X \cup c^{\mathscr{A}}]$ and $\mathscr{H}' = \mathscr{A}[X' \cup c^{\mathscr{A}}]$. In particular, $p_0 \in P_i$ if and only if $p_0' \in P_i'$, where $i \in \{1, 2\}$. By induction over the structure of φ, the claim then easily follows. □

Finally we show that the same holds for *join* nodes of a tree decomposition. Note that the corresponding operation on structures is the union.

Lemma 6 (Join/Union)**.** *Let $\mathscr{A}, \mathscr{B}$ be compatible τ-structures, $X = A \cap B$, and $\varphi \in MSO(\tau)$. Let $\mathcal{G} = \mathcal{EMC}(\mathscr{A}, X, \varphi)$ and $\mathcal{G}' = \mathcal{EMC}(\mathscr{A} \cup \mathscr{B}, X, \varphi)$.*

1. *If $eval(\mathcal{G}) = \top$, then $eval(\mathcal{G}') = \top$.*
2. *If $eval(\mathcal{G}) = \bot$, then $eval(\mathcal{G}') = \bot$.*

Proof. Let $eval(\mathcal{EMC}(\mathscr{A}, X, \varphi)) \in \{\top, \bot\}$. By Corollary 1, $eval(\mathcal{EMC}(\mathscr{A} \cup \mathscr{B}, X \cup (B \setminus A), \varphi)) = eval(\mathcal{EMC}(\mathscr{A}, X, \varphi))$. The claim then immediately follows by Lemma 5. □

3. Reducing the Size of Games

In this section we show that for every game $\mathcal{G} = (P, M, P_0) = \mathcal{EMC}(\mathscr{A}, X, \varphi)$ one can construct a game $\mathcal{G}' = (P', M', p_0)$ such that $eval(\mathcal{G}) = eval(\mathcal{G}')$ if $eval(\mathcal{G}) \in \{\top, \bot\}$, but $P' \subseteq P$ and $M' \subseteq M'$ are typically much smaller than P and M. This will be crucial for obtaining the desired running times of our algorithm. We first define a suitable notion of *equivalence* between games.

Definition 5 (Equivalent Games)**.** We say that two positions p_1, p_2 are *equivalent*, denoted by $p_1 \cong p_2$ iff

- $p_1 = (\mathscr{H}_1, X, \varphi)$ and $p_2 = (\mathscr{H}_2, X, \varphi)$ for some formula φ and set $X \subseteq H_1 \cap H_2$,
- there is an isomorphism $h\colon H_1 \to H_2$ between $\mathscr{H}_1$ and $\mathscr{H}_2$, such that $h(a) = a$ for all $a \in X$.

We say that two games $\mathcal{G}_1 = (P_1, M_1, p_1)$ and $\mathcal{G}_2 = (P_2, M_2, p_2)$ are *equivalent*, denoted by $\mathcal{G}_1 \cong \mathcal{G}_2$, if $p_1 \cong p_2$ and there is a bijection $\pi\colon subgames(\mathcal{G}_1) \to subgames(\mathcal{G}_2)$, such that $\mathcal{G}' \cong \pi(\mathcal{G}')$ for all $\mathcal{G}' \in subgames(\mathcal{G}_1)$.

We now define a *reduce operation* that significantly shrinks the size of a game $\mathcal{G}$ (see Algorithm 3). Firstly, subgames won by the opponent player are removed. If, for instance, the formula is universal, then the falsifier can safely ignore subgames that evaluate as $\top$, i.e., for which the verifier has a winning strategy. For example, it is easy to see that we can remove the two subgames $\top$ and $\bot$ in Figure 1.

Secondly, we only need to keep one representation per equivalence class under $\cong$ for all undetermined games. Here, we use the fact that $eval(\mathcal{G}_1) \cong eval(\mathcal{G}_2)$ for any $\mathcal{G}_1, \mathcal{G}_2$ with $\mathcal{G}_1 \cong \mathcal{G}_2$. We will not explicitly prove this claim. If, however, $\mathcal{G}_1 = \mathcal{EMC}(\mathscr{A}_1, X, \varphi)$ and $\mathcal{G}_2 = \mathcal{EMC}(\mathscr{A}_2, X, \varphi)$ for some τ-structures $\mathscr{A}_1$ and $\mathscr{A}_2$, for $X \subseteq A_1 \cap A_2$ and $\varphi \in \mathrm{MSO}(\tau)$, then the bijection π induced by the definition of $\cong$ yields a bisimulation between $\mathcal{EMC}(\mathscr{A}_1, X, \varphi)$ and $\mathcal{EMC}(\mathscr{A}_2, X, \varphi)$. In particular, if both $\mathcal{G}_1$ and $\mathcal{G}_2$ are subgames of the same game $\mathscr{G}$, then it suffices to keep either subgame as "witness" for possible winning positions for the respective player in the model checking game. Thus,

Algorithm 3 Reducing a game.

Algorithm *reduce*($\mathcal{G}$)
Input: A game $\mathcal{G} = (P, M, p_0)$ with $p_0 = (\mathscr{H}, X, \varphi)$.

if $\mathcal{G} \in \{\top, \bot\}$ **then return** $\mathcal{G}$
if φ is an atomic or negated atomic formula **then return** *eval*($\mathcal{G}$)
Let $P' := \{p_0\}$ and $M' := \emptyset$.
for $p \in next(p_0)$ **do**
    Let $\mathcal{G}' = (P'_1, M'_1, p') := reduce(subgame_{\mathcal{G}}(p))$.
    if φ is universal **and** $\mathcal{G}' = \bot$ **then return** $\bot$
    if φ is existential **and** $\mathcal{G}' = \top$ **then return** $\top$
    if $\mathcal{G}' \notin \{\top, \bot\}$ **and** $\mathcal{G}' \ncong \mathcal{G}''$ for all $\mathcal{G}'' \in subgames((P', M', p_0))$ **then**
        Update $P' := P' \cup P'_1$ and $M' := M' \cup M'_1 \cup \{(p_0, p')\}$.
if $P' = \{p_0\}$ **then return** $eval((P', M', p_0))$.
return (P', M', p_0)

removing equivalent subgames from a game $\mathcal{G}$ can be seen as a variant of taking the *bisimulation quotient* (cf., [45, Chapter 7]) of $\mathcal{G}$.

See Figures 3 and 4 in Section 6 for two examples.

Lemma 7. *Let $\mathscr{A}$ be a τ-structure, $X \subseteq A$, and $\varphi \in MSO(\tau)$. Let $\mathcal{G} = \mathcal{EMC}(\mathscr{A}, X, \varphi)$. Then*

- *$eval(\mathcal{G}) = \top$, if and only if $reduce(\mathcal{G}) = \top$, and*
- *$eval(\mathcal{G}) = \bot$, if and only if $reduce(\mathcal{G}) = \bot$.*

Proof. Let $\mathcal{G} = (P, M, p_0)$, where $p_0 = (\mathscr{H}, X, \varphi)$. Without loss of generality, we assume that $\mathcal{G} \notin \{\top, \bot\}$. We only show the first case ($\top$), the second statement is proven analogously. The proof is an induction over the structure of φ. If φ is an atomic or negated atomic formula or $P = \{p_0\}$, then the statement holds by definition of *reduce*($\mathcal{G}$). For the induction step, assume φ is not an atomic or negated formula, and $next(p_0) \neq \emptyset$.

Let $\mathcal{G}_p = subgame_{\mathcal{G}}(p)$ for all $p \in next(p_0)$ and let $eval(\mathcal{G}) = \top$. If φ is existential, then there is $p \in next(p_0)$ with $eval(\mathcal{G}_p) = \top$. By the induction hypothesis, $reduce(\mathcal{G}_p) = eval(\mathcal{G}_p) = \top$, and therefore $reduce(\mathcal{G}) = \top$. Similarly, if φ is universal, then $eval(\mathcal{G}_p) = \top$ for all $p \in next(p_0)$. By the induction hypothesis, $reduce(\mathcal{G}_p) = \top$ for each $p \in next(p_0)$. Hence, we have $P' = \{p_0\}$ after the for-loop. Since φ is universal, the call to $eval((P', M', p_0))$ returns $\top$ by definition, and therefore $reduce(\mathcal{G}) = \top$.

Conversely, let $reduce(\mathcal{G}) = \top$. If φ is existential, then there must be some $p \in next(p_0)$ with $reduce(\mathcal{G}_p) = \top$. Assume for a contradiction that $reduce(\mathcal{G}_p) = \bot$ for all $p \in next(p_0)$. Then $P' = \{p_0\}$ after the for-loop, which implies $eval((P', M', p_0)) = \bot$, a contradiction. Let therefore p be such a position with $reduce(\mathcal{G}_p) = \top$. Then, by the induction hypothesis, $eval(\mathcal{G}_p) = \top$ for this p, and therefore also $eval(\mathcal{G}) = \top$. If φ is universal, then we know $P' = \{p_0\}$ after the for-loop, as this is the only possibility how $reduce(\mathcal{G})$ can return $\top$.

Therefore, $\mathit{reduce}(\mathcal{G}_p) = \top$ for all $p \in \mathit{next}(p_0)$, and hence $\mathit{eval}(\mathcal{G}) = \top$ by the induction hypothesis and definition of $\mathit{eval}(\mathcal{G})$. □

Now we prove an upper bound for the size of a reduced game. Since this is a general upper bound for arbitrary formulas and structures, we cannot expect better bounds than the known lower bounds (unless P = NP) [2].

Definition 6 (Equivalent Structures)**.** Let τ be a vocabulary and $\varphi \in \mathrm{MSO}(\tau)$. Let $\mathscr{A}_1, \mathscr{A}_2$ be two τ-structures and $X \subseteq A_1 \cap A_2$.

We call $\mathscr{A}_1$ and $\mathscr{A}_2$ *equivalent with respect to* φ *and* X, denoted by $\mathscr{A}_1 \cong_{X,\varphi} \mathscr{A}_2$, if $\mathit{reduce}(\mathcal{EMC}(\mathscr{A}_1, X, \varphi)) \cong \mathit{reduce}(\mathcal{EMC}(\mathscr{A}_2, X, \varphi))$.

For an arbitrary set X of objects, we let

$$\mathcal{STR}(\tau, X) = \{\, \mathscr{A} \in \mathcal{STR}(\tau) \mid X \subseteq A \,\}$$

be the set of all τ-structures that contain X, and $\mathcal{STR}(\tau, X)/\cong_{X,\varphi}$ the set of equivalence classes of $\mathcal{STR}(\tau, X)$ under $\cong_{X,\varphi}$. We let

$$N_{X,\varphi} := |\mathcal{STR}(\tau, X)/\cong_{X,\varphi}|.$$

Lemma 8. *Let τ be a vocabulary, $\varphi \in MSO(\tau)$, and X be a set of objects. Then*

$$N_{X,\varphi} \leq \exp^{qr(\varphi)+1}((|X|+1)^{O(\|\varphi\|)}),$$

where $\|\varphi\|$ is the length of an encoding of φ.

Proof. Without loss of generality, we assume τ is minimal such that $\varphi \in \mathrm{MSO}(\tau)$ and therefore $\|\varphi\| \geq \max\{|\tau|, \mathit{arity}(\tau)\}$. We prove the claim by induction over the structure of φ.

If φ is an atomic or negated atomic formula, let $\bar{c} = \mathit{null}(\tau)$, and $\mathscr{A} \in \mathcal{STR}(\tau, X)$. Let $\mathcal{G}_{\mathscr{A}} = \mathit{reduce}(\mathcal{EMC}(\mathscr{A}, X, \varphi))$. Then either $\mathcal{G}_{\mathscr{A}} \in \{\top, \bot\}$, or $\mathcal{G}_{\mathscr{A}} = (P, M, p_0)$, where $p_0 = (\mathscr{H}, X, \varphi)$ and $\mathscr{H} = \mathscr{A}[X \cup c^{\mathscr{A}}]$. Hence, $N_{X,\varphi}$ depends on the number of non-isomorphic structures on at most $n := |X| + |\bar{c}^{\mathscr{A}}| \leq |X| + |\mathit{null}(\tau)|$ objects. For a fixed relation symbol $R \in \tau$, there are $2^{n^{\mathit{arity}(R)}}$ ways to choose the interpretation $R^{\mathscr{H}}$. The total number of non-isomorphic τ-structures over at most n objects is therefore bounded by $N_{X,\varphi} \leq \exp^{qr(\varphi)+1}((|X|+1)^{O(\|\varphi\|)})$.

If $\varphi = \psi_1 \wedge \psi_2$ or $\varphi = \psi_1 \vee \psi_2$, then $qr(\varphi) = \max\{qr(\psi_1), qr(\psi_2)\}$ and $\|\psi_1\| + \|\psi_2\| \leq \|\varphi\|$. Furthermore, by the induction hypothesis we get $N_{X,\psi_i} \leq \exp^{qr(\varphi)+1}((|X|+1)^{O(\|\psi_i\|)})$. We conclude that $N_{X,\varphi} = O(N_{X,\psi_1} \cdot N_{X,\psi_2}) \leq \exp^{qr(\varphi)+1}((|X|+1)^{O(\|\varphi\|)})$.

If $\varphi \in \{\forall c\psi, \exists c\psi, \forall R\psi, \exists R\psi\}$, then $qr(\psi) = qr(\varphi) - 1$, $\|\psi\| < \|\varphi\|$, and, by the induction hypothesis, $N_{X,\psi} = \exp^{qr(\psi)+1}((|X|+1)^{O(\|\psi\|)})$. Since $\mathit{reduce}()$ ignores equivalent subgames, the total number $N_{X,\varphi}$ is upper-bounded by $2^{N_{X,\psi}} \leq \exp^{qr(\varphi)+1}((|X|+1)^{O(\|\varphi\|)})$. □

Lemma 9. *Let $\mathscr{A}$ be a τ-structure, $X \subseteq A$ and $\varphi \in MSO(\tau)$. Then*

$$|\mathit{reduce}(\mathcal{EMC}(\mathscr{A}, X, \varphi))| \leq \exp^{qr(\varphi)+1}((|X|+1)^{O(\|\varphi\|)}),$$

where $\|\varphi\|$ is the length of an encoding of φ.

Algorithm 4 Combining two games.

Algorithm $combine(\mathcal{G}_1, \mathcal{G}_2)$
Input: Two games $\mathcal{G}_i = (P_i, M_i, p_i)$ with $p_i = (\mathscr{H}_i, X_i, \varphi)$,
where $\mathscr{H}_1$ and $\mathscr{H}_2$ are compatible τ-structures,
$X_i \subseteq H_i$, and $\varphi \in \mathrm{MSO}(\tau)$.

Let $p_0 := (\mathscr{H}_1 \cup \mathscr{H}_2, X_1 \cup X_2, \varphi)$, $P := \{p_0\}$ and $M := \emptyset$.
for each $(p_1', p_2') \in next(p_1) \times next(p_2)$ **do**
  Let $p_1' = (\mathscr{H}_1', X_1, \psi_1)$ and $p_2' = (\mathscr{H}_2', X_2, \psi_2)$.
  if $\psi_1 = \psi_2$ **and** $\mathscr{H}_1'$ and $\mathscr{H}_2'$ are compatible **then**
    Let $(P', M', p_0') = combine(subgame_{\mathcal{G}_1}(p_1'), subgame_{\mathcal{G}_2}(p_2'))$.
    Update $P := P \cup P'$ and $M := M \cup M' \cup \{(p_0, p_0')\}$.
return $reduce((P, M, p_0))$

Proof. We use induction over the structure of φ. If φ is an atomic or negated atomic formula, then $\mathcal{G} = \mathcal{EMC}(\mathscr{A}, X, \varphi)$ contains only a single position and $reduce(\mathcal{G}) \in \{\top, \bot, \mathcal{G}\}$.

If $\varphi = \psi_1 \wedge \psi_2$ or $\varphi = \psi_1 \vee \psi_2$, let, for $i \in \{1, 2\}$, be $\mathcal{G}_{\psi_i} = \mathcal{EMC}(\mathscr{A}, X, \psi_i)$. By the induction hypothesis, $|reduce(\mathcal{G}_{\psi_i})| = \exp^{qr(\varphi)+1}((|X|+1)^{O(\|\psi_i\|)})$ where $qr(\psi_i) \leq qr(\varphi)$ and $\|\psi_1\| + \|\psi_2\| \leq \|\varphi\|$, and therefore,

$$|reduce(\mathcal{G})| \leq 1 + |reduce(\mathcal{G}_{\psi_1})| + |reduce(\mathcal{G}_{\psi_2})| \leq \exp^{qr(\varphi)+1}((|X|+1)^{O(\|\varphi\|)}).$$

If otherwise $\varphi \in \{\forall c\psi, \exists c\psi, \forall R\psi, \exists R\psi\}$, then $qr(\psi) = qr(\varphi) - 1$ and $\|\psi\| < \|\varphi\|$. Since equivalent subgames are ignored,

$$|reduce(\mathcal{G})| \leq 1 + N_{X,\psi} \cdot \exp^{qr(\psi)+1}((|X|+1)^{O(\|\psi\|)}) \leq \exp^{qr(\varphi)+1}((|X|+1)^{O(\|\varphi\|)}).$$

□

4. Combining and Extending Games

In this section, we show how model checking games on structures can be computed inductively. We will introduce two algorithms: Algorithm 4 will be used when structures are *combined*, i.e., taking the union of two compatible structures. This happens at *join* and *introduce* nodes of the tree decomposition. Algorithm 5 will be used when objects are removed from the set X, which happens at *forget* nodes of the tree decomposition. We first will study the case of combining games. The next lemma is required for technical reasons.

Lemma 10. *Let $\mathscr{A}_1$ and $\mathscr{A}_2$ be compatible τ-structures, $\varphi \in MSO(\tau)$ and let $X_1 \subseteq A_1$ and $X_2 \subseteq A_2$ with $A_1 \cap A_2 = X_1 \cap X_2$. Let, for $i \in \{1, 2\}$, $\mathcal{R}_i = reduce(\mathcal{EMC}(\mathscr{A}_i, X_i, \varphi)) \notin \{\top, \bot\}$ and $\mathcal{G}_i = (P_i, M_i, p_i) \cong \mathcal{R}_i$, where $p_i = (\mathscr{H}_i, X_i, \varphi)$. Then $\mathscr{H}_1$ and $\mathscr{H}_2$ are compatible.*

Proof. Let $\bar{c} = \mathit{null}(\tau)$. Since $\mathcal{G}_i \cong \mathcal{R}_i$, we have, by Definition 5, $\mathscr{H}_i \cong \mathscr{A}_i[X_i \cup \bar{c}^{\mathscr{A}_i}]$ for an isomorphism h_i with $h_i(a) = a$ for all $a \in X_i$.

By definition, $\bar{c}^{\mathscr{A}_i} = \{\, c^{\mathscr{A}_i} \mid c \in \bar{c} \cap \mathit{interpreted}(\mathscr{A}_i) \,\}$, and therefore $c \in \mathit{interpreted}(\mathscr{A}_i)$ if and only if $c \in \mathit{interpreted}(\mathscr{H}_i)$.

If $c^{\mathscr{H}_i} \in H_1 \cap H_2$, then in particular $c^{\mathscr{H}_i} \in A_1 \cap A_2 \subseteq X_i$. Hence, $c^{\mathscr{H}_i} = h_i(c^{\mathscr{H}_i}) = c^{\mathscr{A}_i}$. Since $\mathscr{A}_1$ and $\mathscr{A}_2$ are compatible, $c^{\mathscr{A}_1} = c^{\mathscr{A}_2}$ for all $c^{\mathscr{A}_i} \in A_1 \cap A_2$, and therefore $c^{\mathscr{H}_1} = c^{\mathscr{H}_2}$ for all $c^{\mathscr{H}_i} \in H_1 \cap H_2$.

Accordingly, $\mathscr{H}_1$ and $\mathscr{H}_2$ are compatible. □

We now prove that for a structure $\mathscr{A}$ with $\mathscr{A} = \mathscr{A}_1 \cup \mathscr{A}_2$ the reduced model checking game $\mathit{reduce}(\mathcal{EMC}(\mathscr{A}, X, \varphi))$ can, up to equivalence, be computed from $\mathcal{R}_1 = \mathit{reduce}(\mathcal{EMC}(\mathscr{A}_1, X, \varphi))$ and $\mathcal{R}_2 = \mathit{reduce}(\mathcal{EMC}(\mathscr{A}_2, X, \varphi))$. Here, $\mathit{combine}(\mathcal{R}_1, \mathcal{R}_2)$ essentially computes the Cartesian product of plays in the games over $\mathscr{A}_1$ and $\mathscr{A}_2$, respectively. This is possible because each set $U \subseteq A$ can be split into $U \cap A_1$ and $U \cap A_2$, such that $(\mathscr{A}_1, U \cap A_1) \cup (\mathscr{A}_2, U \cap A_2) = (\mathscr{A}, U)$. Similarly, each interpretation of a nullary symbol is either nil, or contained in $A_1 \cap A_2$, in $A_1 \setminus A_2$, or in $A_2 \setminus A_1$ (cf., Figure 2). These cases can be reconstructed from the respective subgames on $\mathscr{A}_1$ and $\mathscr{A}_2$.

Lemma 11. *Let $\mathscr{A}_1$ and $\mathscr{A}_2$ be compatible τ-structures, $\varphi \in \mathit{MSO}(\tau)$ and let $X_1 \subseteq A_1$ and $X_2 \subseteq A_2$ with $A_1 \cap A_2 = X_1 \cap X_2$. Let, for $i \in \{1, 2\}$, $\mathcal{R}_i = \mathit{reduce}(\mathcal{EMC}(\mathscr{A}_i, X_i, \varphi)) \notin \{\top, \bot\}$ and $\mathcal{G}_i \cong \mathcal{R}_i$. Then*

$$\mathit{reduce}(\mathcal{EMC}(\mathscr{A}_1 \cup \mathscr{A}_2, X_1 \cup X_2, \varphi)) \cong \mathit{combine}(\mathcal{G}_1, \mathcal{G}_2).$$

Proof. The proof is an induction over the structure of φ. Let $\mathscr{A} = \mathscr{A}_1 \cup \mathscr{A}_2$, $X = X_1 \cup X_2$, and $\bar{c} = \mathit{null}(\tau)$. Let $\mathcal{R} = (P_\mathcal{R}, M_\mathcal{R}, p_\mathcal{R}) = \mathit{reduce}(\mathcal{EMC}(\mathscr{A}, X, \varphi))$ and $\mathcal{G} = (P_\mathcal{G}, M_\mathcal{G}, p_\mathcal{G}) = \mathit{combine}(\mathcal{G}_1, \mathcal{G}_2)$. Let, for $i \in \{1, 2\}$, $\mathcal{G}_i = (P_{\mathcal{G}_i}, M_{\mathcal{G}_i}, p_{\mathcal{G}_i})$ and $p_{\mathcal{G}_i} = (\mathscr{H}_i, X_i, \varphi)$.

By Lemma 10, $\mathscr{H}_1$ and $\mathscr{H}_2$ are compatible. Furthermore, $\mathscr{A}_i[X_i \cup \bar{c}^{\mathscr{A}_i}] \cong \mathscr{H}_i$, and thus $\mathscr{A}[X \cup \bar{c}^{\mathscr{A}}] = \mathscr{A}_1[X_1 \cup \bar{c}^{\mathscr{A}_1}] \cup \mathscr{A}_2[X_2 \cup \bar{c}^{\mathscr{A}_2}] \cong \mathscr{H}_1 \cup \mathscr{H}_2$.

If $\mathcal{R} \notin \{\top, \bot\}$, then $p_\mathcal{R} = (\mathscr{A}[X \cup \bar{c}^{\mathscr{A}}], X, \varphi)$. Therefore,

$$p_\mathcal{R} = (\mathscr{A}[X \cup \bar{c}^{\mathscr{A}}], X, \varphi) \cong (\mathscr{H}_1 \cup \mathscr{H}_2, X_1 \cup X_2, \varphi) = p_\mathcal{G}.$$

Let φ be an atomic or negated atomic formula. If $\mathcal{R} \notin \{\top, \bot\}$ the lemma already holds with above considerations. Therefore consider the case $\mathcal{R} \in \{\top, \bot\}$, say $\mathcal{R} = \top$. Then $\mathit{eval}(\mathcal{EMC}(\mathscr{A}, X, \varphi)) = \mathcal{R} = \top$ by Lemma 7. Therefore, $\mathcal{R} = \top$ if and only if the verifier wins the play (p_0), where p_0 is the initial position of $\mathcal{EMC}(\mathscr{A}[X \cup \bar{c}^{\mathscr{A}}], X, \varphi)$. The claim then follows, since $p_0 = (\mathscr{A}[X \cup \bar{c}^{\mathscr{A}}], X, \varphi) \cong p_\mathcal{G}$, where in particular $\mathscr{A}[X \cup \bar{c}^{\mathscr{A}}] \cong \mathscr{H}_1 \cup \mathscr{H}_2$ and $X = X_1 \cup X_2$.

For the induction step, we distinguish the following cases.

Case $\varphi = \psi_1 \wedge \psi_2$ or $\varphi = \psi_1 \vee \psi_2$.

Let, for $\psi \in \{\psi_1, \psi_2\}$, $\mathcal{R}_\psi = \mathit{reduce}(\mathcal{EMC}(\mathscr{A}, X, \psi))$ and, for each $i \in \{1, 2\}$, be $\mathcal{R}_{i,\psi} = \mathit{reduce}(\mathcal{EMC}(\mathscr{A}_i, X_i, \psi))$.

Consider $\psi \in \{\psi_1, \psi_2\}$ with $\mathcal{R}_\psi \notin \{\top, \bot\}$ and suppose there was $i \in \{1,2\}$, say $i = 1$, with $\mathcal{R}_{1,\psi} \in \{\top, \bot\}$. Let $\mathcal{U}_{1,\psi} = \mathcal{EMC}(\mathscr{A}_1, X_1, \psi)$ and $\mathcal{U}_\psi = \mathcal{EMC}(\mathscr{A}, X_1 \cup A_2, \psi)$. By Lemma 7, $\mathit{eval}(\mathcal{U}_{1,\psi}) \in \{\top, \bot\}$, and therefore by Corollary 1, $\mathit{eval}(\mathcal{U}_\psi) \in \{\top, \bot\}$. Since $X_1 \cup X_2 \subseteq X_1 \cup A_2$, also $\mathit{eval}(\mathscr{A}, X_1 \cup X_2, \psi) \in \{\top, \bot\}$. This contradicts $\mathcal{R}_\psi \notin \{\top, \bot\}$ via Lemma 7.

Therefore, we have $\mathcal{R}_{i,\psi} \notin \{\top, \bot\}$ for each $i \in \{1,2\}$, which implies $\mathcal{R}_i \notin \{\top, \bot\}$. Since $\mathcal{G}_i \cong \mathcal{R}_i$ for $i \in \{1,2\}$, there is $\mathcal{G}_{i,\psi} \in \mathit{subgames}(\mathcal{G}_i)$ with $\mathcal{G}_{i,\psi} \cong \mathcal{R}_{i,\psi}$. The algorithm $\mathit{combine}(\mathcal{G}_1, \mathcal{G}_2)$ will eventually call $\mathit{combine}(\mathcal{G}_{1,\psi}, \mathcal{G}_{2,\psi})$. Then, by the induction hypothesis, $\mathit{subgames}(\mathit{combine}(\mathcal{G}_1, \mathcal{G}_2))$ contains the required subgame $\mathit{combine}(\mathcal{G}_{1,\psi}, \mathcal{G}_{2,\psi}) \cong \mathcal{R}_\psi$.

Conversely, let $\psi \in \{\psi_1, \psi_2\}$ and $(\mathcal{G}_{1,\psi}, \mathcal{G}_{2,\psi}) \in \mathit{subgames}(\mathcal{G}_1) \times \mathit{subgames}(\mathcal{G}_2)$ such that $\mathit{combine}(\mathcal{G}_1, \mathcal{G}_2)$ recursively calls $\mathit{combine}(\mathcal{G}_{1,\psi}, \mathcal{G}_{2,\psi})$. From $\mathcal{G}_i \cong \mathcal{R}_i$ we get $\mathcal{G}_{i,\psi} \cong \mathcal{R}_{i,\psi}$. Then $\mathit{combine}(\mathcal{G}_{1,\psi}, \mathcal{G}_{2,\psi}) \cong \mathcal{R}_\psi$ by the induction hypothesis.

Together, the statement of the lemma follows.

Case $\varphi = \forall R\psi$ or $\varphi = \exists R\psi$.

Consider an arbitrary $U \subseteq A$ and let $\mathcal{R}' = \mathit{reduce}(\mathcal{EMC}(\mathscr{A}', X, \psi))$, where $\mathscr{A}' = (\mathscr{A}, U)$ with $R^{\mathscr{A}'} = U$. For $i \in \{1,2\}$, let $U_i = U \cap A_i$ and $\mathcal{R}'_i = \mathit{reduce}(\mathcal{EMC}(\mathscr{A}'_i, X_i, \psi))$, where $\mathscr{A}'_i = (\mathscr{A}_i, U_i)$. If $\mathcal{R}' \notin \{\top, \bot\}$, then $\mathcal{R}'_i \notin \{\top, \bot\}$ for each $i \in \{1,2\}$ by using a combination of Lemma 7 and Corollary 1. Therefore, $\mathcal{R}_i \notin \{\top, \bot\}$. Since $\mathcal{G}_i \cong \mathcal{R}_i$, there is $\mathcal{G}'_i = (P'_i, M'_i, p'_i) \in \mathit{subgames}(\mathcal{G}_i)$ with $\mathcal{G}'_i \cong \mathcal{R}'_i$. Let $p'_i = (\mathscr{H}'_i, X_i, \psi)$. Since $\mathscr{A}'_1$ and $\mathscr{A}'_2$ are compatible and $\mathcal{G}'_i \cong \mathcal{R}'_i$, we by Lemma 10 have that $\mathscr{H}'_1$ and $\mathscr{H}'_2$ are compatible. Therefore, the algorithm eventually recursively calls $\mathit{combine}(\mathcal{G}'_1, \mathcal{G}'_2)$. By the induction hypothesis, $\mathit{combine}(\mathcal{G}'_1, \mathcal{G}'_2) \cong \mathcal{R}'$.

Conversely, assume the algorithm recursively calls $\mathit{combine}(\mathcal{G}'_1, \mathcal{G}'_2)$, where $\mathcal{G}'_i = (P'_i, M'_i, p'_i) \in \mathit{subgames}(\mathcal{G}_i)$ for each $i \in \{1,2\}$. From $\mathcal{G}_i \cong \mathcal{R}_i$ we get $\mathcal{G}'_i \cong \mathit{reduce}(\mathcal{EMC}(\mathscr{A}'_i, X_i, \psi))$, where $\mathscr{A}'_i = (\mathscr{A}_i, U_i)$ for some $U_i \subseteq A_i$. Let $p'_i = (\mathscr{H}'_i, X_i, \psi)$. Since $\mathscr{H}_1$ and $\mathscr{H}_2$ are compatible and $A_1 \cap A_2 \subseteq X_i \subseteq H_i$, also $\mathscr{A}'_1$ and $\mathscr{A}'_2$ are compatible. Therefore, the induction hypothesis implies $\mathit{combine}(\mathcal{G}'_1, \mathcal{G}'_2) \cong \mathit{reduce}(\mathcal{EMC}(\mathscr{A}', X, \psi))$, where $\mathscr{A}' = (\mathscr{A}, U_1 \cup U_2)$ with $R^{\mathscr{A}'} = U_1 \cup U_2$.

Together, the statement of the lemma follows.

Case $\varphi = \forall c\psi$ or $\varphi = \exists c\psi$.

Consider a (τ, c)-expansion $\mathscr{A}'$ of $\mathscr{A}$ and let $\mathcal{R}' = \mathit{reduce}(\mathcal{EMC}(\mathscr{A}', X, \psi))$. Let, for $i \in \{1,2\}$, $\mathscr{A}'_i = \mathscr{A}'[A_i]$ be the (τ, c)-expansion of $\mathscr{A}_i$ with $c^{\mathscr{A}'_i} = c^{\mathscr{A}'}$ if $c^{\mathscr{A}'} \in A_i$, and $c^{\mathscr{A}'_i} = \mathrm{nil}$ otherwise. Let $\mathcal{R}'_i = \mathit{reduce}(\mathcal{EMC}(\mathscr{A}'_i, X_i, \psi))$. If $\mathcal{R}' \notin \{\top, \bot\}$, then $\mathcal{R}'_i \notin \{\top, \bot\}$ by a combination of Lemma 7 and Corollary 1. Therefore, $\mathcal{R}_i \notin \{\top, \bot\}$. Since $\mathcal{G}_i \cong \mathcal{R}_i$, there is $\mathcal{G}'_i = (P'_i, M'_i, p'_i) \in \mathit{subgames}(\mathcal{G}_i)$ with $\mathcal{G}'_i \cong \mathcal{R}'_i$. Let $p'_i = (\mathscr{H}'_i, X_i, \psi)$. Since $\mathscr{A}'_1$ and $\mathscr{A}'_2$ are compatible and $\mathcal{G}'_i \cong \mathcal{R}'_i$, Lemma 10 implies that $\mathscr{H}'_1$ and $\mathscr{H}'_2$ are compatible. The algorithm therefore eventually calls $\mathit{combine}(\mathcal{G}'_1, \mathcal{G}'_2)$. By the induction hypothesis, $\mathit{combine}(\mathcal{G}'_1, \mathcal{G}'_2) \cong \mathcal{R}'$.

Algorithm 5 Forgetting an object.

Algorithm $\mathit{forget}(\mathcal{G}, x)$
Input: A game $\mathcal{G} = (P, M, p_0)$ with $p_0 = (\mathscr{H}, X, \varphi)$ and $x \in X$

if there is $c \in \mathit{interpreted}(\mathscr{H})$ with $c^{\mathscr{H}} = x$ **then**
  let $p_0' = (\mathscr{H}, X \setminus \{x\}, \varphi)$
else let $p_0' = (\mathscr{H}[H \setminus \{x\}], X \setminus \{x\}, \varphi)$.
Let $P' = \{p_0'\}$ and $M' = \emptyset$.
for each $\mathcal{G}' \in \mathit{subgames}(\mathcal{G})$ **do**
  Let $(P'', M'', p_0'') = \mathit{forget}(\mathcal{G}')$.
  Set $P' := P' \cup P''$ and $M' := M' \cup M''$.
return $\mathit{reduce}((P', M', p_0'))$

Conversely, assume the algorithm recursively calls $\mathit{combine}(\mathcal{G}_1', \mathcal{G}_2')$, where $\mathcal{G}_i' = (P_i', M_i', p_i') \in \mathit{subgames}(\mathcal{G}_i)$ for each $i \in \{1, 2\}$. From $\mathcal{G}_i \cong \mathcal{R}_i$ we get $\mathcal{G}_i' \cong \mathit{reduce}(\mathscr{A}_i', X_i, \psi)$ for some (τ, c)-expansion $\mathscr{A}_i'$ of $\mathscr{A}_i$. Since $\mathscr{H}_1$ and $\mathscr{H}_2$ are compatible and $A_1 \cap A_2 \subseteq X_i \subseteq H_i$, also $\mathscr{A}_1'$ and $\mathscr{A}_2'$ are compatible. By the induction hypothesis, $\mathit{combine}(\mathcal{G}_1', \mathcal{G}_2') \cong \mathit{reduce}(\mathcal{EMC}(\mathscr{A}', X, \psi))$, where $\mathscr{A}' = \mathscr{A}_1' \cup \mathscr{A}_2'$.

Together, the statement of the lemma follows. □

Lemma 12. *Let $\mathscr{A}$ be a τ-structure, $X \subseteq A$ and $x \in X$. Let $\varphi \in \mathit{MSO}(\tau)$ and $\mathcal{G} \cong \mathit{reduce}(\mathcal{EMC}(\mathscr{A}, X, \varphi)) \notin \{\top, \bot\}$. Then*

$$\mathit{reduce}(\mathcal{EMC}(\mathscr{A}, X \setminus \{x\}, \varphi)) \cong \mathit{forget}(\mathcal{G}, x).$$

Proof. We use induction over the structure of φ. Let $\bar{c} = \mathit{null}(\tau)$, $X' = X \setminus \{x\}$, $\mathcal{R}' = (P_{\mathcal{R}'}, M_{\mathcal{R}'}, p_{\mathcal{R}'}) = \mathit{reduce}(\mathscr{A}, X \setminus \{x\}, \varphi)$. Let $\mathcal{G} = (P_{\mathcal{G}}, F_{\mathcal{G}}, p_{\mathcal{G}})$ with $p_{\mathcal{G}} = (\mathscr{H}, X, \varphi)$. Here, $\mathscr{H} \cong \mathscr{A}[X \cup c^{\mathscr{A}}]$, since $\mathcal{G} \cong \mathit{reduce}(\mathcal{EMC}(\mathscr{A}, X, \varphi))$.

If φ is an atomic or negated atomic formula and $\mathcal{R}' \notin \{\top, \bot\}$, the statement holds since $p_{\mathcal{R}'} = (\mathscr{A}[X' \cup c^{\mathscr{A}}], X', \varphi)$ by definition.

If otherwise φ is an atomic or negated atomic formula and $\mathcal{R}' \in \{\top, \bot\}$, let $\mathscr{H}' = \mathscr{H}[H \setminus \{x\}]$ if $c^{\mathscr{H}} \neq x$ for all $c \in \mathit{interpreted}(\mathscr{H})$, and $\mathscr{H}' = \mathscr{H}$ otherwise. If $\mathcal{R}' \in \{\top, \bot\}$, then $\mathit{eval}(\mathcal{EMC}(\mathscr{A}, X', \varphi)) = \mathcal{R}'$ by Lemma 7. Since $\mathscr{H}' \cong \mathscr{A}[X' \cup c^{\mathscr{A}}]$, we have $\mathit{eval}(\mathcal{EMC}(\mathscr{A}, X', \varphi)) = \mathit{eval}(\mathcal{EMC}(\mathscr{H}', X', \varphi'))$ and

$$\mathit{eval}(\mathcal{EMC}(\mathscr{H}', X', \varphi')) = \mathit{reduce}(\mathcal{EMC}(\mathscr{H}', X', \varphi)) = \mathit{forget}(\mathcal{G}, x).$$

For the induction step, let $\mathcal{G}' \in \mathit{subgames}(\mathcal{G})$ be an arbitrary subgame of $\mathcal{G}$. Since $\mathcal{G} \cong \mathit{reduce}(\mathcal{EMC}(\mathscr{A}, X, \varphi))$, we know that $\mathcal{G}' \cong \mathit{reduce}(\mathcal{EMC}(\mathscr{A}', X, \psi))$ for some expansion $\mathscr{A}'$ of $\mathscr{A}$ and subformula ψ of φ. By the induction hypothesis,

$$\mathit{forget}(\mathcal{G}', x) \cong \mathit{reduce}(\mathcal{EMC}(\mathscr{A}', X \setminus \{x\}, \psi)).$$

Conversely, if $\mathcal{R}'' = \mathit{reduce}(\mathcal{EMC}(\mathscr{A}', X \setminus \{x\}, \psi))$ is a subgame of $\mathcal{R}'$, then $\mathcal{R}'' \notin \{\top, \bot\}$. This implies $\mathit{reduce}(\mathcal{EMC}(\mathscr{A}', X, \psi)) \notin \{\top, \bot\}$ by Lemmas 5

and 7. Therefore, there is $\mathcal{G}' \in \mathit{subgames}(\mathcal{G})$ with $\mathcal{G}' \cong \mathit{reduce}(\mathcal{EMC}(\mathscr{A}', X, \psi))$. By the induction hypothesis, $\mathcal{R}'' \cong \mathit{forget}(\mathcal{G}', x)$.

Together, the statement of the lemma follows. □

Finally, we come back to Algorithm 2 and show that its correctness translates to reduced games.

Lemma 13. *Let $\mathscr{A}$ be a fully interpreted τ-structure, $X \subseteq A$, and $\varphi \in \mathit{MSO}(\tau)$. Let $\mathcal{G} \cong \mathit{reduce}(\mathcal{EMC}(\mathscr{A}, X, \varphi))$. Then*

$$\mathit{eval}(\mathcal{MC}(\mathscr{A}, \varphi)) = \mathit{eval}(\mathit{convert}(\mathcal{G})).$$

Proof. We prove the statement by induction over the structure of φ. Recall that $\mathcal{M} = \mathcal{MC}(\mathscr{A}, X, \varphi)$ is determined and hence $\mathit{eval}(\mathcal{M}) \in \{\top, \bot\}$.

If $\mathcal{G} \in \{\top, \bot\}$, then $\mathcal{G} = \mathit{convert}(\mathcal{G})$. We get $\mathcal{G} = \mathit{eval}(\mathcal{EMC}(\mathscr{A}, X, \varphi))$ from Lemma 7 and therefore, using Lemmma 3 for the first equality,

$$\mathit{eval}(\mathcal{MC}(\mathscr{A}, \varphi)) = \mathit{eval}(\mathcal{EMC}(\mathscr{A}, X, \varphi)) = \mathcal{G} = \mathit{eval}(\mathcal{G}) = \mathit{eval}(\mathit{convert}(\mathcal{G})).$$

Let therefore $\mathcal{G} = (P, M, p_0) \notin \{\top, \bot\}$ with $p_0 = (\mathscr{H}, X, \varphi)$ and suppose $\mathit{eval}(\mathcal{MC}(\mathscr{A}, \varphi)) = \top$ (the case $\bot$ is shown analogously). For atomic or negated atomic formulas, the statement holds since, by definition, $\mathcal{G} \cong \mathit{reduce}(\mathcal{EMC}(\mathscr{A}, X, \varphi)) = \mathcal{EMC}(\mathscr{A}, X, \varphi)$, and hence $\mathcal{MC}(\mathscr{A}, \varphi) = \mathit{convert}(\mathcal{G})$ by Lemma 1.

If $\varphi \in \{\forall R\psi, \exists R\psi\}$, say $\varphi = \forall R\psi$, consider $U \subseteq A$ and let $\mathscr{A}' = (\mathscr{A}, U)$ with $R^{\mathscr{A}'} = U$. If there is $\mathcal{G}' \in \mathit{subgames}(\mathcal{G})$ with $\mathcal{G}' \cong \mathit{reduce}(\mathcal{EMC}(\mathscr{A}', X, \psi))$, then $\mathit{eval}(\mathcal{MC}(\mathscr{A}', \varphi)) = \mathit{eval}(\mathit{convert}(\mathcal{G}'))$ by the induction hypothesis. If otherwise there is no such $\mathcal{G}'$ in $\mathit{subgames}(\mathcal{G})$, then $\mathit{reduce}(\mathcal{EMC}(\mathscr{A}', X, \psi)) = \top$ by definition of $\mathit{reduce}()$, since $\mathcal{G} \notin \{\top, \bot\}$. By Lemmas 7 and 1, we then conclude $\mathit{eval}(\mathcal{MC}(\mathscr{A}', \psi)) = \top$. Together, the lemma follows.

Similarly, if $\varphi \in \{\psi_1 \wedge \psi_2, \psi_1 \vee \psi_2\}$, then for $\psi \in \{\psi_1, \psi_2\}$ either there is $\mathcal{G}' \in \mathit{subgames}(\mathcal{G})$, such that $\mathcal{G}' \cong \mathit{reduce}(\mathscr{A}, X, \psi)$, or there is no such $\mathcal{G}'$ contained in $\mathit{subgames}(\mathcal{G})$. In the former case we again obtain $\mathit{eval}(\mathcal{MC}, \psi) = \mathit{eval}(\mathit{convert}(\mathcal{G}'))$ by the induction hypothesis, and in the latter case we can again argue that $\mathcal{G}' \cong \mathit{reduce}(\mathcal{EMC}(\mathscr{A}, X, \psi)) \in \{\top, \bot\}$.

Finally, let $\varphi \in \{\forall c\psi, \exists c\psi\}$. For any $a \in A$ and $\mathscr{A}' = (\mathscr{A}, a)$, where $c^{\mathscr{A}'} = a$, we argue analogously to the previous cases that either there is $\mathcal{G}' \in \mathit{subgames}(\mathcal{G})$, such that $\mathcal{G}' \cong \mathit{reduce}(\mathscr{A}', X, \psi)$, or there is no such $\mathcal{G}'$ contained in $\mathit{subgames}(\mathcal{G})$, which implies $\mathcal{G}' \cong \mathit{reduce}(\mathcal{EMC}(\mathscr{A}', X, \psi)) \in \{\top, \bot\}$.

Hence, consider the (τ, c)-expansion $\mathscr{A}'$ of $\mathscr{A}$ with $c^{\mathscr{A}'} = \mathrm{nil}$. If there is $\mathcal{G}' = (P', M', p_0') \in \mathit{subgames}(\mathcal{G})$ with $\mathcal{G}' \cong \mathit{reduce}(\mathcal{EMC}(\mathscr{A}', X, \psi))$, then $\mathcal{G}' \notin \{\top, \bot\}$. In particular, $\mathcal{G}' = (\mathscr{H}', X, \psi)$, where $\mathscr{H}'$ is not fully interpreted. Therefore, $\mathit{convert}(\mathcal{G})$ removes the subgame $\mathcal{G}'$ from $\mathcal{G}$. In either case, $\mathit{convert}(\mathcal{G})$ does only contain subgames where c has been interpreted as an object in A, as considered above. Together, the statement of the lemma then follows. □

5. Courcelle's Theorem

We can now reprove Courcelle's Theorem for LinMSO-definable optimization problems. Throughout this section, we shall abbreviate $\mathit{reduce}(\mathscr{A}, X, \varphi) := \mathit{reduce}(\mathcal{EMC}(\mathscr{A}, X, \varphi))$.

Theorem 2. *Fix a relational vocabulary τ, a set $\bar{R} = \{R_1, \ldots, R_l\} \subseteq \mathit{unary}(\tau)$ of unary relation symbols, and $\tau' = \tau \setminus \bar{R}$. Let $\varphi \in \mathit{MSO}(\tau)$, and $w, \alpha_1, \ldots, \alpha_l \in \mathbf{Z}$ be constants. Given a τ'-structure $\mathscr{A}$ together with a tree decomposition $(\mathcal{T}, \mathcal{X})$ of $\mathscr{A}$ having width at most w, where $\mathcal{T} = (T, F)$ and $\mathcal{X} = (X_i)_{i \in T}$, one can compute*

$$\min\left\{ \sum_{k=1}^{l} \alpha_k |U_k| \;\middle|\; U_i \subseteq A,\ 1 \leq i \leq l,\ \textit{and}\ (\mathscr{A}, U_1, \ldots, U_l) \models \varphi \right\}$$

in time $O(|T|)$.

The remainder of this section is devoted to the proof of this theorem. We give an algorithm that essentially works as follows: In a first phase, the algorithm uses dynamic programming on the tree decomposition (based on Lemmas 14–17) to compute the reduced extended model checking games $\mathcal{G} \cong \mathit{reduce}(\mathscr{A}_i', \emptyset, \varphi)$ and the values $\sum_{k=1}^{l} \alpha_k |U_k|$ for *all* structures $\mathscr{A}_i' = (\mathscr{A}, U_1, \ldots, U_l)$ where $U_i \subseteq A$ for $1 \leq i \leq l$. Note that by the previous sections the algorithm does not need to distinguish between equivalent games. In a second phase, the algorithm tests whether the verifier has a winning strategy on $\mathit{convert}(G)$, or, in other words (Lemma 13), whether $(\mathscr{A}, U_1, \ldots, U_l) \models \varphi$. The algorithm then collects the values $\sum_{k=1}^{l} \alpha_k |U_k|$ for all $\mathscr{A}_i' = (\mathscr{A}, U_1, \ldots, U_l)$ with $\mathscr{A}_i' \models \varphi$ and outputs the optimal one. Since most of the games considered are equivalent (Lemma 8), we can obtain the desired run time bounds.

Without loss of generality, we assume $X_{\mathit{root}(\mathcal{T})} = \emptyset$. Recall that for each $i \in T$, $\mathscr{A}_i$ is the substructure of $\mathscr{A}$ induced by those objects that appear at or below i in the tree decomposition. Let, for $i \in T$,

$$\mathcal{AR}_i = \mathscr{P}(A_i) \times \cdots \times \mathscr{P}(A_i) = \mathscr{P}(A_i)^l$$

be the set of possible interpretations of the free relation symbols $(R_1, \ldots, R_l)$ in $\mathscr{A}_i$,

$$\mathcal{EXP}_i = \{\, (\mathscr{A}_i, U_1, \ldots, U_l) \mid (U_1, \ldots, U_l) \in \mathcal{AR}_i \,\}$$

be the set of their corresponding τ-expansions of $\mathscr{A}_i$, where for each $1 \leq j \leq l$ the symbol R_j is interpreted as U_j, and

$$\mathcal{RED}_i = \{\, \mathit{reduce}(\mathscr{A}_i', X_i, \varphi) \mid \mathscr{A}_i' \in \mathcal{EXP}_i \,\}$$

be the corresponding extended model checking games in their reduced form. We let $(U_1, \ldots, U_l) \cap X_i := (U_1 \cap X_i, \ldots, U_l \cap X_i)$ and

$$\mathcal{AR}_i \cap X_i = \{\, (U_1, \ldots, U_l) \cap X_i \mid (U_1, \ldots, U_l) \in \mathcal{AR}_i \,\}$$

be the restriction of $\mathcal{AR}_i$ to X_i, and let, for $\bar{U} = (U_1, \ldots, U_l) \in \mathcal{AR}_i \cap X_i$,

$$\mathcal{EXP}_i(\bar{U}) = \{\, \mathscr{A}_i' \in \mathcal{EXP}_i \mid R_j^{\mathscr{A}_i'} \cap X_i = U_j \text{ for } 1 \leq j \leq l \,\}$$

be the set of τ-expansions of $\mathscr{A}$ that "match" $\bar{U}$ on X_i. Let

$$\mathcal{RED}_i(\bar{U}) = \{\, \mathit{reduce}(\mathscr{A}_i', X_i, \varphi) \mid \mathscr{A}_i' \in \mathcal{EXP}_i(\bar{U}) \,\}$$

be the corresponding games, and, for arbitrary games $\mathcal{R}$,

$$\mathcal{EXP}_i(\bar{U}, \mathcal{R}) = \{\, \mathscr{A}_i' \in \mathcal{EXP}_i(\bar{U}) \mid \mathcal{R} \cong \mathit{reduce}(\mathscr{A}_i', X_i, \varphi) \,\}$$

and

$$\mathcal{RED}_i(\bar{U}, \mathcal{R}) = \{\, \mathcal{R}' \in \mathcal{RED}_i(\bar{U}) \mid \mathcal{R} \cong \mathcal{R}' \,\}.$$

Finally, we let, for $\bar{U} = (U_1, \ldots, U_l) \in \mathcal{AR}_i$,

$$\mathscr{A}_i(\bar{U}) = (\mathscr{A}_i, U_1, \ldots, U_l)[X_i],$$

where $R_i^{\mathscr{A}_i(\bar{U})} = U_i \cap X_i$ for each $1 \leq i \leq l$, and

$$\mathcal{R}(\bar{U}) = \mathit{reduce}(\mathscr{A}_i(\bar{U}), X_i, \varphi).$$

5.1. The Algorithm

We use dynamic programming on the tree decomposition as follows. As usual, we associate with each node $i \in T$ of the tree decomposition a *table* S_i that contains feasible, *partial* solutions and their corresponding *value* val_i under the optimization function.

Formally, we let $S_i \colon \mathcal{AR}_i \cap X_i \to \mathscr{P}(\mathcal{RED}_i \setminus \{\bot\})$ map tuples $\bar{U} \in \mathcal{AR}_i \cap X_i$ to sets of *feasible* games over $\mathscr{A}_i$, i.e., games $\mathcal{R}$ with $\mathcal{R} \neq \bot$, and let $\mathit{val}_i \colon \mathcal{RED}_i \to \mathbf{Z}_\infty$ be the corresponding values, where $\mathbf{Z}_\infty = \mathbf{Z} \cup \{\infty\}$.

Initially, we let $S_i(\bar{U}) := \emptyset$ for all $\bar{U} \in \mathcal{AR}_i \cap X_i$ and $\mathit{val}_i(\mathcal{R}) := \infty$ for all $\mathcal{R} \in \mathcal{RED}_i$.

Phase 1. The algorithm traverses the tree decomposition bottom-up. Recall that each node $i \in T$ is either a leaf, or of one of the three types *introduce*, *forget*, or *join*. The algorithm distinguishes these four cases as follows.

leaf Let $X_i = \{x\}$. For all $\bar{U} = (U_1, \ldots, U_l) \in \mathcal{AR}_i \cap X_i$ the algorithm considers $\mathcal{R}(\bar{U}) = \mathit{reduce}(\mathscr{A}_i(\bar{U}), X_i, \varphi)$. If $\mathcal{R}(\bar{U}) \neq \bot$, then the algorithm sets

$$S_i(\bar{U}) := \{\mathcal{R}(\bar{U})\} \qquad \text{and} \qquad \mathit{val}_i(\mathcal{R}(\bar{U})) := 0.$$

introduce Let j be the unique child of i and $X_i = X_j \cup \{x\}$ for $x \notin A_j$.

For each $\bar{U}_j = (U_{j,1}, \ldots, U_{j,l}) \in \mathcal{AR}_j \cap X_j$, and each $\bar{U}_i = (U_{i,1}, \ldots, U_{i,l}) \in \mathcal{AR}_i \cap X_i$ such that $(U_{j,1}, \ldots, U_{j,l}) = (U_{i,1} \cap X_j, \ldots, U_{i,l} \cap X_j)$, the algorithm considers each $\mathcal{R}_j \in S_j(\bar{U}_j)$.

Let

$$\mathcal{R}_i = \begin{cases} \top & \text{if } \mathcal{R}_j = \top \text{ and} \\ \mathit{combine}(\mathcal{R}_j, \mathcal{R}(\bar{U}_i)) & \text{otherwise.} \end{cases}$$

If there is $\mathcal{R}'_i \in S_i(\bar{U}_i)$ with $\mathcal{R}'_i \cong \mathcal{R}_i$, then let $\mathcal{R}_i := \mathcal{R}'_i$ instead.

If $\mathcal{R}_i \neq \bot$, the algorithm sets

$$S_i(\bar{U}) := S_i(\bar{U}) \cup \{\mathcal{R}_i\} \qquad \text{and} \qquad \mathit{val}_i(\mathcal{R}_i) := \min\{\mathit{val}_i(\mathcal{R}_i), \mathit{val}_j(\mathcal{R}_j)\}.$$

forget Let j be the unique child of i and $X_i \cup \{x\} = X_j$ for $x \notin A_i$.

For each $\bar{U}_j = (U_{j,1}, \ldots, U_{j,l}) \in \mathcal{AR}_j \cap X_j$ the algorithm considers each $\mathcal{R}_j \in S_j(\bar{U}_j)$. Let $\bar{U}_i = (U_{i,1}, \ldots, U_{i,l}) = (U_{j,1} \cap X_i, \ldots, U_{j,l} \cap X_i)$ and

$$\mathcal{R}_i = \begin{cases} \top & \text{if } \mathcal{R}_j = \top \text{ and} \\ \mathit{forget}(\mathcal{R}_j, x) & \text{otherwise.} \end{cases}$$

If there is $\mathcal{R}'_i \in S_i(\bar{U}_i)$ with $\mathcal{R}'_i \cong \mathcal{R}_i$, then let $\mathcal{R}_i := \mathcal{R}'_i$ instead. If now $\mathcal{R}_i \neq \bot$, the algorithm sets $S_i(\bar{U}_i) := S_i(\bar{U}_i) \cup \{\mathcal{R}_i\}$ and

$$\mathit{val}_i(\mathcal{R}_i) := \min\left\{\mathit{val}_i(\mathcal{R}_i), \mathit{val}_j(\mathcal{R}_j) + \sum_{k=1}^{l} \alpha_k(x \in U_{i,k})\right\}.$$

where $(x \in U_{j,k}) \in \{0, 1\}$ as defined in Section 1.

join Let j_1, j_2 be the children of i. Then $X_i = X_{j_1} = X_{j_2}$.

For each $\bar{U} = (U_1, \ldots, U_l) \in \mathcal{AR}_i \cap X_i$ the algorithm considers each pair $(\mathcal{R}_{j_1}, \mathcal{R}_{j_2}) \in S_{j_1}(\bar{U}) \times S_{j_2}(\bar{U})$. Let

$$\mathcal{R}_i = \begin{cases} \top & \text{if } \mathcal{R}_{j_1} = \top \text{ or } \mathcal{R}_{j_2} = \top \text{ and} \\ \mathit{combine}(\mathcal{R}_{j_1}, \mathcal{R}_{j_2}) & \text{otherwise.} \end{cases}$$

If there is $\mathcal{R}'_i \in S_i(\bar{U}_i)$ with $\mathcal{R}'_i \cong \mathcal{R}_i$, then let $\mathcal{R}_i := \mathcal{R}'_i$ instead. If now $\mathcal{R}_i \neq \bot$, the algorithm sets $S_i(\bar{U}_i) := S_i(\bar{U}_i) \cup \{\mathcal{R}_i\}$ and

$$\mathit{val}_i(\mathcal{R}_i) := \min\left\{\mathit{val}_i(\mathcal{R}_i), \mathit{val}_{j_1}(\mathcal{R}_{j_1}) + \mathit{val}_{j_2}(\mathcal{R}_{j_2})\right\}.$$

Phase 2. Let $r = \mathit{root}(\mathcal{T})$ and

$$\bar{U}_r = (\emptyset, \ldots, \emptyset) \in \mathcal{AR}_r \cap X_r = \mathcal{AR}_r \cap \emptyset.$$

The algorithm starts with $OPT := \infty$ and considers each $\mathcal{R}_r \in S_r(\bar{U}_r)$. If $\mathit{eval}(\mathit{convert}(\mathcal{R}_r)) = \top$, then the algorithm updates

$$OPT := \min\{OPT, \mathit{val}_r(\mathcal{R}_r)\}.$$

Finally, the algorithm outputs OPT.

5.2. Proofs

In order to show that the algorithm is correct and computes the optimal solution, we use induction over the structure of the tree decomposition to show the following invariant.

Invariant 1. *After the algorithm has processed a node $i \in T$ in Phase 1, for each $\bar{U} = (U_1, \ldots, U_l) \in \mathcal{AR}_i \cap X_i$ we have that*

(I) for each $\mathscr{A}'_i \in \mathcal{EXP}_i(\bar{U})$ with $\mathcal{R} = reduce(\mathscr{A}'_i, X_i, \varphi) \neq \bot$ there is exactly one $\mathcal{R}' \in S_i(\bar{U})$ with $\mathcal{R}' \cong \mathcal{R}$,

(II) for each game $\mathcal{R} \in S_i(\bar{U})$ we have $\mathcal{R} \neq \bot$ and $\mathcal{RED}_i(\bar{U}, \mathcal{R}) \neq \emptyset$, and

(III) for each $\mathcal{R} \in S_i(\bar{U})$ we have

$$val_i(\mathcal{R}) = \min\left\{ \sum_{k=1}^{l} \alpha_k |R_k^{\mathscr{A}'_i} \setminus X_i| \;\middle|\; \mathscr{A}'_i \in \mathcal{EXP}_i(\bar{U}, \mathcal{R}) \wedge reduce(\mathscr{A}'_i, X_i, \varphi) \neq \bot \right\}.$$

Here, (I) guarantees that S_i is *complete*, i.e., $S_i(\bar{U})$ contains games for all feasible partial solutions, (II) guarantees that all games in $S_i(\bar{U})$ do, in fact, correspond to a reduced game over some τ-expansion of $\mathscr{A}_i$, and (III) guarantees that we also compute the correct solution, i.e., $val_i(\mathcal{R})$ is optimal for $\mathcal{RED}_i(\bar{U}, \mathcal{R})$. Note that the "exactly one" in (I) is required for the claimed running time, but not for the correctness of the solution.

Lemma 14. *Invariant 1 holds for leafs of the tree decomposition.*

Proof. Let $i \in T$ be a leaf and $\bar{U} = (U_1, \ldots, U_l) \in \mathcal{AR}_i \cap X_i = \mathcal{AR}_i$. Since i is a leaf, we have

$$\mathcal{RED}_i(\bar{U}) = \{\, \mathscr{A}_i(\bar{U}) \mid \bar{U} \in \mathcal{AR}_i \cap X_i \,\},$$

such that (I) and (II) clearly hold. Furthermore, $\mathcal{R}_j^{\mathscr{A}_i(\bar{U})} \setminus X_i = \emptyset$ for all $1 \leq j \leq l$, since $A_i \setminus X_i = \emptyset$, and therefore $val_i(\mathcal{R}) = 0$ for all $\mathcal{R} \in \mathcal{RED}_i$. □

Lemma 15. *Let $i \in T$ be an introduce node of the tree decomposition and $j \in T$ be the unique child of i. If Invariant 1 holds for j before the algorithm processes i, then it also holds for i.*

Proof. Let $X_i = X_j \cup \{x\}$, where $x \notin A_j$. Let $\bar{U}_i = (U_{i,1}, \ldots, U_{i,l}) \in \mathcal{AR}_i \cap X_i$ and $\bar{U}_j = (U_{j,1}, \ldots, U_{j,l}) \in \mathcal{AR}_j \cap X_j$ with $(U_{j,1}, \ldots, U_{j,l}) = (U_{i,1} \cap X_j, \ldots, U_{i,l} \cap X_j)$.

Consider $\mathscr{A}'_i \in \mathcal{EXP}_i(\bar{U}_i)$ and let $\mathscr{A}'_j = \mathscr{A}'_i[A_j]$. If $\mathcal{R}_i = reduce(\mathscr{A}'_i, X_i, \varphi) \neq \bot$, then also $\mathcal{R}_j = reduce(\mathscr{A}'_j, X_j, \varphi) \neq \bot$ by Lemma 4 and Lemma 7. By Invariant 1, $S_j(\bar{U}_j)$ therefore contains exactly one game $\mathcal{R}'_j$ with $\mathcal{R}'_j \cong \mathcal{R}_j$. If $\mathcal{R}'_j = \top$, then $\mathcal{R}_i = \top$ by Lemma 4 and Lemma 7. Otherwise, the algorithm

computes $\mathcal{R}'_i = \mathit{combine}(\mathcal{R}'_j, \mathcal{R}(\bar{U}_i))$. By Lemma 11, $\mathcal{R}'_i \cong \mathcal{R}_i$, which implies part (I) of the invariant.

Conversely, consider $\mathcal{R}_i \in S_i(\bar{U}_i)$. Then either $\mathcal{R}_i = \top$ and there is $\mathcal{R}_j \in S_j(\bar{U}_j)$ with $\mathcal{R}_j = \top$, or there is $\mathcal{R}_j \in S_j(\bar{U}_j)$ with $\mathcal{R}_i \cong \mathit{combine}(\mathcal{R}_j, \mathcal{R}_i(\bar{U}_i))$. By the invariant for j, $\mathcal{RED}_j(\mathcal{R}_j) \neq \emptyset$. From this we get there is $\mathcal{R}'_j \in \mathcal{RED}_j(\bar{U}_j, \mathcal{R}_j)$ such that $\mathcal{R}'_j \cong \mathcal{R}_j$ and $\mathcal{R}'_j = \mathit{reduce}(\mathscr{A}'_j, X_j, \varphi)$ for some $\mathscr{A}'_j \in \mathcal{EXP}_j(\bar{U}_j)$. Let $\mathscr{A}'_i \in \mathcal{EXP}_i(\bar{U}_i)$, chosen in a way such that $(R_1^{\mathscr{A}'_j}, \ldots, R_l^{\mathscr{A}'_j}) = (R_1^{\mathscr{A}'_i} \cap A_j, \ldots, R_l^{\mathscr{A}'_i} \cap A_j)$.

If $\mathcal{R}_j = \top$, then, by Lemma 4 and Lemma 7, $\mathit{reduce}(\mathscr{A}'_i, X_i, \varphi) = \top \in S_i(\bar{U}_i)$. Otherwise, $\mathit{reduce}(\mathscr{A}_i, X_i, \varphi) \cong \mathit{combine}(\mathcal{R}_j, \mathcal{R}(\bar{U}_i))$ by Lemma 11. Either case implies (II).

Finally, let $\mathcal{R}_i \in S_i(\bar{U}_i)$ and $\mathscr{O}_i \in \mathcal{EXP}_i(\bar{U}_i, \mathcal{R}_i)$ with $\mathit{reduce}(\mathscr{O}_i, X_i, \varphi) \neq \bot$ and

$$\sum_{k=1}^{l} \alpha_k |R_k^{\mathscr{O}_i} \setminus X_i| = \min\left\{ \sum_{k=1}^{l} \alpha_k |R_k^{\mathscr{A}'_i} \setminus X_i| \;\middle|\; \mathscr{A}'_i \in \mathcal{EXP}_i(\bar{U}_i, \mathcal{R}_i) \wedge \mathit{reduce}(\mathscr{A}'_i, X_i, \varphi) \neq \bot \right\}.$$

Let $\mathscr{O}_j = \mathscr{O}_i[A_j]$. By Lemmas 4 and 7, $\mathcal{R}_j = \mathit{reduce}(\mathscr{O}_j, X_j, \varphi) \neq \bot$. Therefore, either $\mathcal{R}_j = \mathit{reduce}(\mathscr{O}_j, X_j, \varphi) = \mathcal{R}_i = \top$, or otherwise $\mathit{combine}(\mathcal{R}_j, \mathcal{R}(\bar{U}_i)) \cong \mathit{reduce}(\mathscr{O}_j \cup \mathscr{A}_i(\bar{U}_i), X_i, \varphi) \cong \mathcal{R}_i$ by Lemma 11.

We need that $\mathscr{O}_j$ is optimal for $\mathcal{EXP}_j(\bar{U}_j, \mathcal{R}_j)$. To this end, assume there was $\mathscr{A}'_j \in \mathcal{EXP}_j(\bar{U}_j, \mathcal{R}_j)$ with $\mathcal{R}'_j = \mathit{reduce}(\mathscr{A}'_j, X_j, \varphi)$, such that either $\mathcal{R}'_j = \top$ or $\mathcal{R}_i \cong \mathit{combine}(\mathcal{R}'_j, \mathscr{A}_i(\bar{U}_i))$, and furthermore

$$\sum_{k=1}^{l} \alpha_k |R_k^{\mathscr{O}_j} \setminus X_j| > \sum_{k=1}^{l} \alpha_k |R_k^{\mathscr{A}'_j} \setminus X_j|.$$

Since, $\mathcal{R}'_j \cong \mathcal{R}_j$, we have, by Lemma 11,

$$\mathcal{R}'_i \cong \begin{cases} \top & \text{if } \mathcal{R}_j = \top \text{ and} \\ \mathit{combine}(\mathcal{R}_j, \mathcal{R}(\bar{U}_i)) & \text{otherwise,} \end{cases}$$

where $\mathcal{R}'_i = \mathit{reduce}(\mathscr{A}'_i, X_i, \varphi)$ and $\mathscr{A}'_i = \mathscr{A}'_j \cup \mathscr{A}_i(\bar{U}_i)$. Therefore,

$$\sum_{k=1}^{l} \alpha_k |R_k^{\mathscr{O}_i} \setminus X_i| > \sum_{k=1}^{l} \alpha_k |R_k^{\mathscr{A}_i} \setminus X_i|,$$

a contradiction to the minimality of $\mathscr{O}_i$. We conclude that $\mathscr{O}_j$ is optimal for $\mathcal{EXP}_j(\bar{U}_j, \mathcal{R}_j)$. From this we get that

$$\mathit{val}_j(\mathcal{R}_j) = \sum_{k=1}^{l} \alpha_k |R_k^{\mathscr{O}_j} \setminus X_j|$$

by the invariant for j, which implies (III). □

Lemma 16. *Let $i \in T$ be a forget node of the tree decomposition and $j \in T$ be the unique child of i. If Invariant 1 holds for j before the algorithm processes i, then it also holds for i.*

Proof. Let j be the unique child of i and $X_i \cup \{x\} = X_j$ for $x \notin X_i$. Note that $\mathscr{A}_i = \mathscr{A}_j$. Let $\bar{U}_i = (U_{i,1}, \ldots, U_{i,l}) \in \mathcal{AR}_i \cap X_i$.

Consider $\mathscr{A}'_i \in \mathcal{EXP}_i(\bar{U}_i)$ with $\mathcal{R}_i = \mathit{reduce}(\mathscr{A}'_i, X_i, \varphi) \neq \bot$ and let $\bar{U}_j = (U_{j,1}, \ldots, U_{j,l}) = (R_1^{\mathscr{A}'_i}, \ldots, R_l^{\mathscr{A}'_i}) \cap X_j \in \mathcal{AR}_j \cap X_j$ and $\mathcal{R}_j = \mathit{reduce}(\mathscr{A}'_i, X_j, \varphi)$. Then, by Lemma 4 and Lemma 7, $\mathcal{R}_j \neq \bot$. Therefore, by the invariant for j, there is $\mathcal{R}'_j \in S_j(\bar{U}_j)$ with $\mathcal{R}'_j \cong \mathcal{R}_j$. If $\mathcal{R}'_j = \mathcal{R}_j = \top$, then, by Lemma 5, also $\mathcal{R}_i = \top$. Otherwise, the algorithm computes $\mathcal{R}'_i = \mathit{forget}(\mathcal{R}'_j, x) \cong \mathcal{R}_j$. Either case implies (I).

Conversely, consider $\mathcal{R}_i \in S_i(\bar{U}_i)$. Then either $\mathcal{R}_i = \top$ and there is $\bar{U}_j \in \mathcal{AR}_j \cap X_j$ and $\mathcal{R}_j \in S_j(\bar{U}_j)$ with $\mathcal{R}_j = \top$ and $\bar{U}_i = \bar{U}_j \cap X_i$, or there is $\bar{U}_j \in \mathcal{AR}_j \cap X_j$ and $\mathcal{R}_j \in S_j(\bar{U}_j)$, such that $\bar{U}_i = \bar{U}_j \cap X_i$ and $\mathcal{R}_i \cong \mathit{forget}(\mathcal{R}_j, x)$. By the invariant for j, in either case $\mathcal{RED}_j(\mathcal{R}_j) \neq \emptyset$. Therefore, there is $\mathcal{R}'_j \in \mathcal{RED}_j(\bar{U}_j, \mathcal{R})$, where $\mathcal{R}'_j \cong \mathcal{R}_j$ and $\mathcal{R}'_j = \mathit{reduce}(\mathscr{A}'_j, X_j, \varphi)$, for some $\mathscr{A}'_j \in \mathcal{EXP}_j(\bar{U}_j)$.

Let $\mathscr{A}'_i = \mathscr{A}'_j$. If $\mathcal{R}_j = \top$, then, by Lemmas 5 and 7, $\mathit{reduce}(\mathscr{A}'_i, X_i, \varphi) = \top \in S_i(\bar{U}_i)$. Otherwise, $\mathit{reduce}(\mathscr{A}'_i, X_i, \varphi) \cong \mathit{forget}(\mathcal{R}_j, x) \cong \mathcal{R}_i$ according to Lemma 12. Either case implies (II).

Finally, consider $\mathcal{R}_i \in S_i(\bar{U}_i)$ and let $\mathscr{O}_i \in \mathcal{EXP}_i(\bar{U}_i, \mathcal{R}_i)$ such that

$$\sum_{k=1}^{l} \alpha_k |R_k^{\mathscr{O}_i} \setminus X_i| = \min\left\{ \sum_{k=1}^{l} \alpha_k |R_k^{\mathscr{A}'_i} \setminus X_i| \;\middle|\; \mathscr{A}'_i \in \mathcal{EXP}_i(\bar{U}_i, \mathcal{R}_i) \wedge \mathit{reduce}(\mathscr{A}'_i, X_i, \varphi) \neq \bot \right\}$$

and $\mathit{reduce}(\mathscr{O}_i, X_i, \varphi) \neq \bot$. Let $\mathscr{O}_j = \mathscr{O}_i$. Then, by Lemmas 5 and 7, $\mathcal{R}_j = \mathit{reduce}(\mathscr{O}_j, X_j, \varphi) \neq \bot$. By (II), there is $\mathcal{R}'_j \in S_j(\bar{U}_j)$ with $\mathcal{R}'_j \cong \mathcal{R}_j$, where $\bar{U}_j = (R_1^{\mathscr{O}_j} \cap X_j, \ldots, R_l^{\mathscr{O}_j} \cap X_j)$. Analogue to the previous case, we obtain that $\mathscr{O}_j$ is optimal in $\mathcal{RED}_i(\bar{U}_j, \mathcal{R}_j)$. Therefore, by the induction hypothesis,

$$\mathit{val}_j(\mathcal{R}_j) = \sum_{k=1}^{l} \alpha_k |R_k^{\mathscr{O}_j} \setminus X_j| = \sum_{k=1}^{l} \alpha_k |R_k^{\mathscr{O}_j} \setminus X_i| - \sum_{k=1}^{l} \alpha_k (x \in R_k^{\mathscr{O}_i} \setminus X_i),$$

which implies (III).

□

Lemma 17. *Let $i \in T$ be a join node of the tree decomposition with children $j_1, j_2 \in T$. If Invariant 1 holds for j_1 and j_2 before the algorithm processes i, then it also holds for i.*

Proof. Note that $X_i = X_{j_1} = X_{j_2}$. Let $\bar{U} = (U_1, \ldots, U_l) \in \mathcal{AR}_i \cap X_i = \mathcal{AR}_{j_1} \cap X_{j_1} = \mathcal{AR}_{j_2} \cap X_{j_2}$.

Consider $\mathscr{A}_i' \in \mathcal{EXP}_i(\bar{U})$ and let, for $j \in \{j_1, j_2\}$, be $\mathscr{A}_j' = \mathscr{A}_i'[A_j]$. If $\mathcal{R}_i = \mathit{reduce}(\mathscr{A}_i', X_i, \varphi) \neq \bot$, then, for $j \in \{j_1, j_2\}$, also $\mathcal{R}_j = \mathit{reduce}(\mathscr{A}_j', X_j, \varphi) \neq \bot$ by Lemma 6 and Lemma 7. By the invariant for $j \in \{j_1, j_2\}$, $S_j(\bar{U})$ therefore contains exactly one $\mathcal{R}_j'$ with $\mathcal{R}_j' \cong \mathcal{R}_j$. If $\mathcal{R}_j' = \top$, then $\mathcal{R}_i = \top$ by Lemma 6 and Lemma 7. Otherwise, the algorithm computes $\mathcal{R}_i' = \mathit{combine}(\mathcal{R}_{j_1}, \mathcal{R}_{j_2})$. By Lemma 11, $\mathcal{R}_i' \cong \mathcal{R}_i$, which implies (I).

Conversely, consider $\mathcal{R}_i \in S_i(\bar{U})$. Then either $\mathcal{R}_i = \top$ and there is $j \in \{j_1, j_2\}$ and $\mathcal{R}_j \in S_j(\bar{U})$ with $\mathcal{R}_j = \top$, or there is $(\mathcal{R}_{j_1}, \mathcal{R}_{j_2}) \in S_{j_1}(\bar{U}) \times S_{j_2}(\bar{U})$, such that $\mathcal{R}_i \cong \mathit{combine}(\mathcal{R}_{j_1}, \mathcal{R}_{j_2})$. By the invariant for $j \in \{j_1, j_2\}$, we have $\mathcal{RED}_j(\mathcal{R}_j) \neq \emptyset$, and therefore there is $R_j' \in \mathcal{RED}_j(\bar{U}, \mathcal{R}_j)$ with $\mathcal{R}_j' \cong \mathcal{R}_j$ and $\mathcal{R}_j' = \mathit{reduce}(\mathscr{A}_j', X_j, \varphi)$, where $\mathscr{A}_j' = (\mathscr{A}_j, R_1^{\mathscr{A}_j}, \ldots, R_l^{\mathscr{A}_j}) \in \mathcal{EXP}_j(\bar{U})$. Let $\mathscr{A}_i' = (\mathscr{A}_i, R_1^{\mathscr{A}_i}, \ldots, R_l^{\mathscr{A}_i}) \in \mathcal{EXP}_i(\bar{U})$, such that $R_k^{\mathscr{A}_i} = R_k^{\mathscr{A}_{j_1}} \cup R_k^{\mathscr{A}_{j_2}}$ for all $1 \leq k \leq l$.

If $\top \in \{\mathcal{R}_{j_1}, \mathcal{R}_{j_2}\}$, then, by Lemmas 6 and 7, $\mathit{reduce}(\mathscr{A}_i', X_i, \varphi) = \top \in S_i(\bar{U})$. Otherwise, $\mathit{reduce}(\mathscr{A}_i, X_i, \varphi) \cong \mathit{combine}(\mathcal{R}_{j_1}, \mathcal{R}_{j_2})$ by Lemma 11. Either case implies (II).

Now consider $\mathcal{R}_i \in S_i(\bar{U})$ and $\mathscr{O}_i \in \mathcal{EXP}_i(\bar{U}, \mathcal{R}_i)$ with $\mathit{reduce}(\mathscr{O}_i, X_i, \varphi) \neq \bot$ and

$$\sum_{k=1}^{l} \alpha_k |R_k^{\mathscr{O}_i} \setminus X_i| = \min\left\{ \sum_{k=1}^{l} \alpha_k |R_k^{\mathscr{A}_i'} \setminus X_i| \;\middle|\; \mathscr{A}_i' \in \mathcal{EXP}_i(\bar{U}, \mathcal{R}_i) \wedge \mathit{reduce}(\mathscr{A}_i', X_i, \varphi) \neq \bot \right\}.$$

Let, for $j \in \{j_1, j_2\}$, $\mathscr{O}_j = \mathscr{O}_i[A_j]$. Then, by Lemma 6 and Lemma 7, $\mathcal{R}_j = \mathit{reduce}(\mathscr{O}_j, X_j, \varphi) \neq \bot$. Therefore, either $\mathcal{R}_j = \mathit{reduce}(\mathscr{O}_j, X_j, \varphi) = \mathcal{R}_i = \top$ for some $j \in \{j_1, j_2\}$, or $\mathit{combine}(\mathcal{R}_{j_1}, \mathcal{R}_{j_2}) \cong \mathit{reduce}(\mathscr{O}_{j_1} \cup \mathscr{O}_{j_2}, X_i, \varphi) \cong \mathcal{R}_i$ by Lemma 11.

Assume there were $j \in \{j_1, j_2\}$, say $j = j_1$, and $\mathscr{A}_j' \in \mathcal{EXP}_j(\bar{U}, \mathcal{R}_j)$ with $\mathcal{R}_j' = \mathit{reduce}(\mathscr{A}_j', X_j, \varphi)$, such that

$$\sum_{k=1}^{l} \alpha_k |R_k^{\mathscr{O}_j} \setminus X_j| > \sum_{k=1}^{l} \alpha_k |R_k^{\mathscr{A}_j'} \setminus X_j|$$

and either $\mathcal{R}_j' = \top$ or $\mathcal{R}_i \cong \mathit{combine}(\mathcal{R}_j', \mathcal{R}_{j_2}')$ for some $\mathcal{R}_{j_2}' \in \mathcal{RED}_{j_2}(\bar{U}, \mathcal{R}_{j_2})$. Since $A_{j_1} \cap A_{j_2} = X_i$, structures $(\mathscr{A}_{j_1}', \mathscr{A}_{j_2}') \in \mathcal{RED}_{j_1}(\bar{U}) \times \mathcal{RED}_{j_2}(\bar{U})$ are compatible. By the invariant, part (II), we have $\mathcal{R}_{j_2}' \cong \mathit{reduce}(\mathscr{A}_{j_2}', X_{j_2}, \varphi)$ for some $\mathscr{A}_{j_2}' \in \mathcal{EXP}_{j_2}(\bar{U})$. Without loss of generality, we assume $\mathscr{A}_{j_2}' = \mathscr{O}_{j_2}$, since each $\mathscr{A}_{j_2}'$ with

$$\sum_{k=1}^{l} \alpha_k |R_k^{\mathscr{O}_{j_2}} \setminus X_j| \geq \sum_{k=1}^{l} \alpha_k |R_k^{\mathscr{A}_{j_2}'} \setminus X_j|$$

yields the same contradiction. Therefore, since $\mathcal{R}'_j \cong \mathcal{R}_j$, we have

$$\mathcal{R}'_i \cong \begin{cases} \top & \text{if } \mathcal{R}'_j = \top \text{ or } \mathcal{R}_{j_2} = \top \text{ and} \\ \mathit{combine}(\mathcal{R}'_j, \mathcal{R}_{j_2}) & \text{otherwise,} \end{cases}$$

by Lemma 11, where $\mathcal{R}'_i = \mathit{reduce}(\mathscr{A}'_j \cup \mathscr{O}_{j_2}, X_i, \varphi)$. Therefore,

$$\sum_{k=1}^{l} \alpha_k |R_k^{\mathscr{O}_i} \setminus X_i| > \sum_{k=1}^{l} \alpha_k |R_k^{\mathscr{A}'_j} \setminus X_i| + \sum_{k=1}^{l} \alpha_k |R_k^{\mathscr{O}_{j_2}} \setminus X_i|$$

a contradiction to the minimality of $\mathscr{O}_i$. Therefore, for $j \in \{j_1, j_2\}$, $\mathscr{O}_j$ is optimal in $\mathcal{EXP}_i(\bar{U}, \mathcal{R}_j)$, and

$$\mathit{val}_j(\mathcal{R}_j) = \sum_{k=1}^{l} \alpha_k |R_k^{\mathscr{O}_j} \setminus X_j|$$

by the invariant for j. By (II), there is $\mathcal{R}'_j \in S_j(\bar{U})$ with $\mathcal{R}'_j \cong \mathcal{R}_j$, which then implies (III). □

Lemma 18. *Let $r = \mathit{root}(\mathcal{T})$ be the root of the tree decomposition, where $X_r = \emptyset$, and let Invariant 1 hold for r. Let $\bar{U} = (\emptyset, \ldots, \emptyset)$ and*

$$\mathit{OPT} = \min\left\{ \sum_{k=1}^{l} \alpha_k |U_k| \;\middle|\; U_i \subseteq A,\ 1 \leq i \leq l, \text{ and } (\mathscr{A}, U_1, \ldots, U_l) \models \varphi \right\}$$

be an optimal solution for the LinMSO-problem. Then

$$\mathit{OPT} = \min\{\, \mathit{val}_r(\mathcal{R}) \mid \mathcal{R} \in S_r(\bar{U}) \wedge \mathit{eval}(\mathit{convert}(\mathcal{R})) = \top \,\}.$$

Proof. Note that $\mathscr{A} = \mathscr{A}_r$. Let $\mathscr{A}'$ be optimal, i.e., let $\mathscr{A}'$ be a τ-expansion of $\mathscr{A}$, such that $\mathscr{A}' \models \varphi$ and

$$\sum_{k=1}^{l} \alpha_k |R_k^{\mathscr{A}'} \setminus X_r| = \sum_{k=1}^{l} \alpha_k |R_k^{\mathscr{A}'}| = \mathit{OPT}.$$

Let $\mathcal{R} = \mathit{reduce}(\mathscr{A}', X_r, \varphi)$. We have $\mathit{eval}(\mathcal{MC}(\mathscr{A}', \varphi)) = \top$ since $\mathscr{A}' \models \varphi$, and therefore

$$\mathit{eval}(\mathit{convert}(\mathcal{R})) = \mathit{eval}(\mathcal{MC}(\mathscr{A}', \varphi)) = \top$$

by Lemma 13. Note that $X_r = \emptyset$ and therefore $\mathcal{AR}_r \cap X_r = \{(\emptyset, \ldots, \emptyset)\}$. By Invariant 1, part (I), there is $\mathcal{R}' \in S_j(\bar{U})$, such that $\mathcal{R}' \cong \mathcal{R}$, which implies $\mathit{OPT} = \mathit{val}_r(\mathcal{R}')$ by part (III) and the optimality of $\mathscr{A}'$ for $\mathcal{EXP}_r(\bar{U}, \mathcal{R})$. Since $\mathit{eval}(\mathit{convert}(\mathcal{R}')) = \top$, we also have

$$\mathit{OPT} = \mathit{val}_r(\mathcal{R}') \geq \min\{\, \mathit{val}_r(\mathcal{R}'') \mid \mathcal{R}'' \in S_r(\bar{U}) \wedge \mathit{eval}(\mathit{convert}(\mathcal{R}'')) = \top \,\}.$$

Conversely, let $\mathcal{R} \in S_r(\bar{U})$, such that $\mathit{eval}(\mathit{convert}(\mathcal{R})) = \top$ and

$$\mathit{val}_r(\mathcal{R}) = \min\{\, \mathit{val}_r(\mathcal{R}') \mid \mathcal{R}' \in S_r(\bar{U}) \wedge \mathit{eval}(\mathit{convert}(\mathcal{R}')) = \top \,\}.$$

By part (II) of the invariant, there is a τ-expansion $\mathscr{A}''$ of $\mathscr{A}$, such that $\mathcal{R} \cong \mathit{reduce}(\mathscr{A}'', X_r, \varphi)$. Since $\mathit{eval}(\mathit{convert}(\mathcal{R})) = \top$, we have $\mathscr{A}'' \models \varphi$ by Lemma 13. Without loss of generality, we can assume by part (III), that $\mathscr{A}''$ is optimal for $\mathcal{EXP}_r(\bar{U}, \mathcal{R})$, i.e., $\mathit{val}_r(\mathcal{R}) = \sum_{k=1}^{l} \alpha_k |R_k^{\mathscr{A}''} \setminus X_r|$. We then directly conclude

$$\mathit{val}_r(\mathcal{R}) = \sum_{k=1}^{l} \alpha_k |R_k^{\mathscr{A}''}| \geq \mathit{OPT} = \sum_{k=1}^{l} \alpha_k |R_k^{\mathscr{A}'}|.$$

□

We can now prove Theorem 2.

Proof of Theorem 2. Using induction over the structure of the tree decomposition and Lemmas 14–17 for the respective nodes, we know that Invariant 1 holds for the root node of the tree decomposition after the algorithm has finished Phase 1. By Lemma 18, the algorithm outputs the correct solution in Phase 2.

For the running time, consider $i \in T$. We have $|\mathcal{AR}_i \cap X_i| = O(2^{|X_i| l})$, which for constant $l \leq |\tau|$ and $|X_i| \leq w + 1$ is a constant. For $\bar{U} \in \mathcal{AR}_i \cap X_i$, consider the set $S_i(\bar{U})$. Since the algorithm only inserts games into $S_i(\bar{U})$, if $S_i(\bar{U})$ does not already contain an equivalent game,

$$|S_i(\bar{U})| \leq N_{X_i,\varphi} \leq \exp^{qr(\varphi)+1}((|X_i| + 1)^{O(\|\varphi\|)}),$$

by Lemma 8, which for bounded $|X_i|$ is constant. Furthermore, by Lemma 9, for each $\mathcal{R} \in S_i(\bar{U})$,

$$|\mathcal{R}| \leq \exp^{qr(\varphi)+1}((|X_i| + 1)^{O(\|\varphi\|)}),$$

again a constant. Finally, each position of each game is of the form $(\mathscr{H}, X_i, \psi)$, where $\|\psi\| \leq \|\varphi\|$ and $\|\mathscr{H}\| = O(|X_i| + \|\varphi\|)$, where $\|\mathscr{H}\|$ denotes the size of a suitable encoding of $\mathscr{H}$. All operations on games, i.e., *reduce*(), *eval*(), *combine*(), *forget*(), and *convert*(), therefore take constant time.

In total, at a node $i \in T$, a constant number of entries or pairs, respectively, is considered, and each operation takes constant time. The running time is therefore $O(|T|)$. □

5.3. Extensions

Semiring Homomorphisms. Note that the algorithm implicitly used a homomorphism

$$h\colon (U_1, \ldots, U_l) \mapsto \sum_{k=1}^{l} \alpha_k |U_k|$$

from the semiring $(\mathscr{P}(\mathcal{AR}_r), \hat{\uplus}, \cup, \hat{\emptyset}, \emptyset)$ into the semiring $(\mathbf{Z}_\infty, +, \min, 0, \infty)$. Here, $\mathscr{P}(\mathcal{AR}_r)$ is the set of all possible interpretations of the free relation symbols (i.e., a set of tuples of sets), $\hat{\uplus}$ is a component-wise, disjoint union with neutral element $\hat{\emptyset} = (\emptyset, \ldots, \emptyset)$, and $\cup$ is the regular union of sets. The extension to other semiring homomorphisms, e.g., to count the number of interpretations satisfying the MSO property φ, is rather straightforward. See [4] for a list of many interesting semirings.

Many-sorted Structures. In this article, we considered one-sorted structures, i.e., structures whose universe contains objects of a single sort only. The corresponding theory is also called MS_1-theory in the literature and is strictly less powerful than corresponding logics for multi-sorted structures. For instance, recall from Example 1 that a graph $G = (V, E)$ can in a natural way be identified with a structure over the vocabulary $\tau_{\mathrm{Graph}} = (adj)$, where V is identified with the one-sorted universe of *vertices*, and adj is interpreted as E. The Hamiltonian Path problem for graphs cannot be expressed in $\mathrm{MSO}(\tau_{\mathrm{Graph}})$, since this requires the use of edge-set quantification (see [35], for instance).

Fortunately, this poses no restriction in algorithmic applications. Firstly, it is not hard to extend the techniques in this paper to many-sorted structures. Courcelle's original works [1, 5] were already proven for many-sorted structures. Secondly, one can easily simulate many-sorted structures by introducing relation symbols that distinguish the respective objects in a common universe accordingly. For example, one can consider the incidence graph of a graph and introduce unary relation symbols V and E, which allow to distinguish objects of sort "vertex" or "edge", and a new binary relation symbol *inc* for the incidence relation. Transforming a structure and a corresponding tree decomposition accordingly can be done efficiently and does not increase the width of the decomposition. Graphs with multi-edges can be represented similarly.

6. Solving Concrete Problems

In the analysis of the running time of the algorithm, we were rather pessimistic w.r.t. the constants hidden in the $O(|T|)$. Recall that unless P = NP, these cannot be bounded by an elementary function, i.e., the running time of the algorithm cannot be $O(f(\|\varphi\|, w)n)$ for a fixed function $f\colon \mathbf{N} \times \mathbf{N} \to \mathbf{N}$ that is a nesting of exponentials of bounded depth [2].

The picture changes dramatically once we assume the *problem* is fixed, i.e., the problem description consisting of the vocabulary τ, a formula $\varphi \in \mathrm{MSO}(\tau)$ and the integers $\alpha_1, \ldots, \alpha_l \in \mathbf{Z}$ are constants. Specialized and comparably efficient algorithms exist for many problems, e.g., of running time $O(2^w\, poly(w)n)$ for the Minimum Vertex Cover problem, or of $O(3^w\, poly(w)n)$ for Minimum Dominating Set and 3-Colorability, cf. [46, 47], where $poly(w)$ is a fixed polynomial in w. Recent results furthermore indicate that better running times are improbable [48]. Assuming small treewidth, such algorithms might still turn out to be feasible in many practical applications, cf. [43].

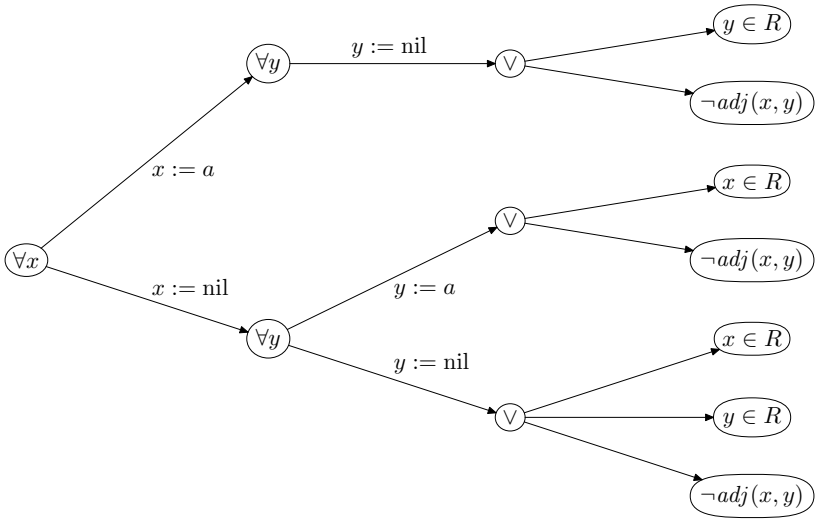

Figure 3: Simplified schematic of $\mathit{reduce}(\mathscr{A}, X, vc)$, where $\mathscr{A}$ has universe $A = \{a\}$, $X = A$ and $a \notin R^{\mathscr{A}}$. If any of the symbols x or y remains uninterpreted (cases $x := \text{nil}$ and $y := \text{nil}$ in the figure), then some of the plays in $\mathcal{EMC}(\mathscr{A}, \emptyset, vc)$ end with a draw and still persist in the reduced game. If $\mathscr{A} = \mathscr{A}_i$ and $X = X_i$ for a node i of a tree decomposition, then this essentially means that it is still open whether nodes in the "future" of i will be adjacent or whether they will be contained in R.

In this section, we estimate the running times of our generic approach for the three aforementioned problems. Let $(\mathcal{T}, \mathcal{X})$ be a tree decomposition of the input graph structure $\mathscr{A}$ over τ_{Graph}, where $\mathcal{T} = (T, F)$ and $\mathcal{X} = (X_i)_{i \in T}$ with $|X_i| \leq w$ for all $i \in T$, i.e., $\mathscr{A}$ has treewidth at most $w - 1$.

6.1. Minimum Vertex Cover

Recall from Example 2 that the formula

$$vc = \forall x \forall y (\neg adj(x, y) \vee x \in R \vee y \in R) \in \text{MSO}(\tau_{\text{Graph}} \cup \{R\})$$

is true on a (τ_{Graph}, R)-structure $(\mathscr{G}, U)$ if and only if $U \subseteq G$ is a vertex cover for the graph $\mathscr{G}$. Using the notation from the previous section, we claim that for each $i \in T$ and for all $\bar{U} \in \mathcal{AR}_i \cap X_i$, the set $S_i(\bar{U})$ contains at most one entry $\mathcal{R}$, and if $\mathcal{R} \in S_i(\bar{U})$ for some $\bar{U}$, then $|\mathcal{R}| = \mathit{poly}(w)$. To this end, consider arbitrary $\bar{U} \in \mathcal{AR}_i \cap X_i$ and let $\mathscr{A}_i' \in \mathcal{EXP}_i(\bar{U})$.

For any $a \in A_i$, such that $a \in R^{\mathscr{A}_i'}$, the verifier has a winning strategy on $\mathcal{G} = \mathcal{EMC}(\mathscr{A}_i'', X_i, \forall y \ldots)$, where $\mathscr{A}_i'' = (\mathscr{A}_i', a)$ with $x^{\mathscr{A}_i''} = a$, since the atomic formula $x \in R$ is always satisfied for all y. Therefore, $\mathit{eval}(\mathcal{G}) = \mathit{reduce}(\mathcal{G}) = \top$ and $\mathit{reduce}()$ removes the subgame $\mathcal{G}$ from $\mathcal{EMC}(\mathscr{A}_i', X_i, vc)$.

Consider now a subgame $\mathcal{EMC}(\mathscr{A}_i'', X_i, \forall y \ldots)$, where $\mathscr{A}_i'' = (\mathscr{A}_i', a)$ with $a \notin R^{\mathscr{A}_i''}$. If there is $b \in A_i$, such that $(a, b) \in \mathit{adj}^{\mathscr{A}_i''}$ and $b \notin R^{\mathscr{A}_i''}$,

then the falsifier has a winning strategy on $\mathcal{EMC}(\mathscr{A}_i', X_i, vc)$ and consequently $\mathit{reduce}(\mathscr{A}_i', X_i, vc) = \bot$. If otherwise for all $b \in A_i$ either $b \in R^{\mathscr{A}_i''}$ or $(a,b) \notin \mathit{adj}^{\mathscr{A}_i''}$, then we get $\mathit{reduce}((\mathscr{A}_i', a, b), X_i, \ldots)) = \top$, and the corresponding subgame will be removed by $\mathit{reduce}()$. Therefore only the subgame on $\mathscr{A}_i''$ with $y^{\mathscr{A}_i''} = \text{nil}$ remains undetermined. We conclude $\mathcal{EMC}((\mathscr{A}_i', b_1), X_i, \forall y \ldots) \cong \mathcal{EMC}((\mathscr{A}_i', b_2), X_i, \forall y \ldots)$ for all $b_1, b_2 \in A_i \setminus X_i$.

Due to the symmetry of x and y in the vertex cover formula, we can argue analogously for the cases where the roles of x and y have been interchanged. Therefore, $\mathcal{R}_1 \cong \mathcal{R}_2$ for all $\mathcal{R}_1, \mathcal{R}_2 \in \mathcal{RED}_i(\bar{U})$, from which we conclude $|S_i(\bar{U})| \leq 1$. Each game is of size $|\mathcal{R}| = O(w)$, since by above considerations

$$|\mathit{subgames}(\mathit{reduce}(\mathscr{A}_i'', X_i, \forall y \ldots))| \leq \begin{cases} |X_i| + 1 + 1 & \text{if } x^{\mathscr{A}_i''} = \text{nil} \\ 1 & \text{if } x^{\mathscr{A}_i''} \in A_i \end{cases}$$

and $|\mathit{subgames}(\mathit{reduce}(\mathscr{A}_i', X_i, vc))| \leq |X_i| + 1 + 1$: In both cases, we have $|X_i|$ subgames for the vertices in X_i, one subgame for all vertices in $A_i \setminus X_i$ (since all of them are equivalent), and one subgame for the case that x and y, respectively, remain uninterpreted. See Figure 3 for an example.

It is not hard to see that $\mathit{reduce}(\mathcal{R})$, $\mathit{eval}(\mathcal{R})$, $\mathit{convert}(\mathcal{R})$, $\mathit{forget}(\mathcal{R}_1)$ and $\mathit{combine}(\mathcal{R}_1, \mathcal{R}_2)$ can be implemented in a way such that they run in time polynomial in $|\mathcal{R}|$ and $|\mathcal{R}_1| + |\mathcal{R}_2|$. Hence, we immediately find that the generic algorithm introduced in this article reaches, up to factors polynomial in w, the running time of $O(2^w n)$ of the specialized algorithm, since $|\mathcal{AR}_i \cap X_i| = 2^{|X_i|}$ for all $i \in T$.

6.2. Minimum Dominating Set

The formula

$$ds = \forall x(x \in R \vee \exists y(y \in R \wedge \mathit{adj}(x,y))) \in \mathrm{MSO}(\tau_{\mathrm{Graph}} \cup \{R\})$$

holds in $(\mathscr{G}, U)$ if and only if $U \subseteq G$ is a dominating set for the graph $\mathscr{G}$. Let for each $i \in T$ and $\bar{U} = (U_1) \in \mathcal{AR}_i \cap X_i$ be $k = |X_i| - |U_1|$. We claim that $|S_j(\bar{U})| \leq 2^k$. To this end, let again $\mathscr{A}_i' \in \mathcal{EXP}_i(\bar{U})$ and $\mathcal{R} = \mathit{reduce}(\mathscr{A}_i', X_i, ds)$. Let $U \subseteq A_i$ be such that $\mathscr{A}_i' = (\mathscr{A}_i, U)$.

If U dominates $a \in A_i$, then either $a \in U$ and $\mathit{reduce}((\mathscr{A}_i, a), X_i, x \in R) = \top$, or there is $b \in U$ that is adjacent to a, and $\mathit{reduce}((\mathscr{A}_i, a), X_i, \exists y \ldots) = \top$. In both cases we get $\mathcal{R}' = \mathit{reduce}((\mathscr{A}_i, a), X_i, x \in R \vee \exists y \ldots) = \top$, and therefore $\mathcal{R}' \notin \mathit{subgames}(\mathcal{R})$.

If $a \in A_i$ is not dominated by U, then $\mathit{reduce}((\mathscr{A}_i', a), X_i, x \in R) = \bot$ and $\mathit{reduce}(\mathscr{A}_i'', X_i, y \in R \wedge \mathit{adj}(x,y)) = \bot$ for all $\mathscr{A}_i''$ with $x^{\mathscr{A}_i''} = a$ and $y^{\mathscr{A}_i''} \in A_i$. These games are therefore removed by $\mathit{reduce}()$. Only the game $\mathit{reduce}(\mathscr{A}_i'', X_i, y \in R \wedge \mathit{adj}(x,y)$ with $x^{\mathscr{A}_i''} = a$ and $y^{\mathscr{A}_i''} = \text{nil}$ remains undetermined. Thus for all $a_1, a_2 \in A_i \setminus X_i$ that are not dominated by U we have $\mathit{reduce}((\mathscr{A}_i', a_1), X_i, x \in R \vee \exists y \ldots) \cong \mathit{reduce}((\mathscr{A}_i', a_2), X_i, x \in R \vee \exists y \ldots)$.

For $\mathscr{A}_i''$ with $x^{\mathscr{A}_i''} = \text{nil}$ the game $\mathit{reduce}(\mathscr{A}_i'', X_i, x \in R)$ remains undetermined. For all $b \in A_i \setminus U$ we have $\mathit{reduce}((\mathscr{A}_i'', b), X_i, y \in R \wedge \mathit{adj}(x,y)) = \bot$

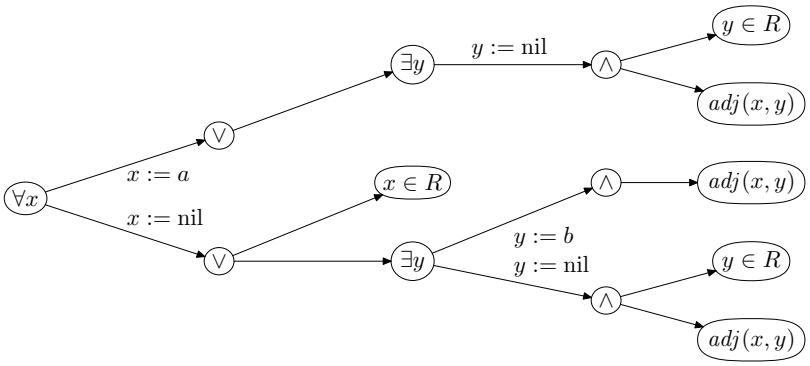

Figure 4: Simplified schematic of $\mathit{reduce}(\mathscr{A}, X, \mathit{ds})$, where $\mathscr{A}$ has universe $A = \{a, b\}$, $X = A$, and $R^{\mathscr{A}} = \{b\}$, such that a and b are not adjacent. Then a might still be dominated by a "future" vertex; the corresponding plays (following the upper $y := \text{nil}$ branch in the figure) end with a draw and therefore persist in the reduced game. Similarly, the branch $x := \text{nil}$ corresponds to the case that "future" vertices are chosen as interpretations for x. Such vertices can also be dominated by b, which is represented by the $y := b$ branch in the figure.

due to the subformula $y \in R$; the corresponding subgame is therefore removed from $\mathcal{EMC}(\mathscr{A}_i'', X_i, \exists y \ldots)$. For all $b_1, b_2 \in A_i \cap U$ we again have $\mathit{reduce}((\mathscr{A}_i'', b_1), X_i, y \in R \wedge \mathit{adj}(x, y)) \cong \mathit{reduce}((\mathscr{A}_i'', b_2), X_i, y \in R \wedge \mathit{adj}(x, y))$.

All in all, either two games $\mathcal{R}_1, \mathcal{R}_2 \in \mathcal{RED}_i(\bar{U})$ only differ w.r.t. the subset of *undominated* nodes in X_i. Since there are k nodes in X_i that are not contained in U, this bounds $|S_j(\bar{U})| \leq 2^k$. For each of them, we have $|\mathit{subgames}(\mathit{reduce}(\mathscr{A}_i'', X_i, \forall x \ldots))| \leq |X_i| + 1 + 1$ corresponding to at most $|X_i|$ undominated nodes in X_i, at most one undominated node in $A_i \setminus X_i$ and the subgame for $\mathscr{A}_i''$ with $x^{\mathscr{A}_i''} = \text{nil}$. Furthermore, $|\mathit{subgames}(\mathit{reduce}(\mathscr{A}_i'', X_i, x \in R \vee \exists y \ldots))| = O(1)$ for $\mathscr{A}_i''$ with $x^{\mathscr{A}_i''} \neq \text{nil}$ and $|\mathit{subgames}(\mathit{reduce}(\mathscr{A}_i'', X_i, x \in R \vee \exists y \ldots))| \leq |X_i| + 1 + 1$. We conclude that $|\mathcal{R}| = O(|X_i|)$. See Figure 4 for an example.

In total, at a node $i \in T$, there are therefore at most

$$\sum_{k=0}^{w} \binom{w}{k} 2^k = 3^w$$

entries stored, and each entry has size $|\mathcal{R}| = O(w)$. Nodes $i \in T$ of type *leaf*, *forget* are therefore processed in time $O(3^w \mathit{poly}(w))$. For *join* nodes $i \in T$ with children j_1, j_2, every pair in $S_{j_1}(\bar{U}) \times S_{j_2}(\bar{U})$ is considered. Therefore, at most

$$\sum_{\bar{U} \in \mathcal{AR}_i \cap X_i} |S_{j_1}(\bar{U})| \cdot |S_{j_2}(\bar{U})| \leq \sum_{k=0}^{w} \binom{w}{k} 2^k 2^k = 5^w$$

entries are considered, which yields a running time of $O(5^w \mathit{poly}(w) n)$. This does not yet match the best specialized algorithm for the Minimum Dominating

SET problem [47] with a running time of $O(3^w poly(w)n)$, but is still faster than combining all pairs with a running time of $\Theta(9^w poly(w)n)$. We note that both the $O(3^w poly(w)n)$ bound from [47] and the $O(4^w n)$ bound from [49] exploit a certain "monotonicity" property of domination like problems, which does not hold for all problems that are expressible in MSO (INDEPENDENT DOMINATING SET being an example).

6.3. 3-COLORABILITY

The formula

$$\mathit{3col} = \exists R_1 \exists R_2 \exists R_3 \Bigg(\forall x \Big(\bigvee_{i=1}^{3} (x \in R_i) \wedge \bigwedge_{i \neq j} (\neg x \in R_i \vee \neg x \in R_j) \Big) \wedge \forall x \forall y \Big(\neg \mathit{adj}(x,y) \vee \bigwedge_{i=1}^{3} (\neg x \in R_i \vee \neg y \in R_i) \Big) \Bigg) \in \mathrm{MSO}(\tau_{\mathrm{Graph}})$$

defining the 3-COLORABILITY problem has no free symbols. Therefore $\mathcal{AR}_i = \{()\}$, where $()$ is the empty tuple, and the table S_j contains at most one entry $\mathcal{R} = \mathit{reduce}(\mathscr{A}_i, X_i, \mathit{3col})$. We estimate the size of $\mathcal{R}$. For, let $\mathit{3col} = \exists R_1 \exists R_2 \exists R_3 \varphi$, where $\varphi = \mathit{part} \wedge \mathit{is}$. Here, $\mathit{part} = \forall x \ldots$ expresses that the R_i are a partition of the universe, and $\mathit{is} = \forall x \forall y \ldots$ ensures that each R_j is an independent set.

If $\bar{U} = (U_1, U_2, U_3) \in \mathscr{P}(A_i)^3$ is not a partition of A_i, then the falsifier wins $\mathcal{EMC}((\mathscr{A}_i, U_1, U_2, U_3), X_i, \mathit{part})$, and therefore $\mathit{reduce}((\mathscr{A}_i, \bar{U}), X_i, \varphi) \notin \mathit{subgames}(\mathcal{R})$. Otherwise, $\mathcal{EMC}((\mathscr{A}_i, \bar{U}, a), X_i, \mathit{part}) = \top$ for all $a \in A_i$ and undetermined when x remains uninterpreted. Using the same arguments as for the similar vertex cover formula vc, we have $\mathcal{R}_1 \cong \mathcal{R}_2$ for all $\bar{U}_j = (U_{j,1}, U_{j,2}, U_{j,3}) \in \mathscr{P}(A_i)^3$ with $\bar{U}_1 \cap X_i = \bar{U}_2 \cap X_i$ and $\mathcal{R}_j = \mathit{reduce}((\mathscr{A}_i, \bar{U}_j, X_i, \mathit{is}) \neq \bot$, $1 \leq j \leq 2$. This implies $\mathit{reduce}((\mathscr{A}_i, \bar{U}_1), X_i, \varphi) \cong \mathit{reduce}((\mathscr{A}_i, \bar{U}_2), X_i, \varphi)$. Thus, $\mathit{subgames}(\mathcal{R})$ contains at most $O(3^w)$ subgames $\mathcal{R}_i = \mathit{reduce}((\mathscr{A}_i, \bar{U}), X_i, \ldots) \neq \bot$, which bounds $|\mathcal{R}| = O(3^w poly(w))$.

Thus, assuming $\mathit{combine}(\mathcal{R}_1, \mathcal{R}_2)$ requires time $\Theta(|\mathcal{R}_1| \cdot |\mathcal{R}_2| \cdot (\|\varphi\| + |X_i|))$, we only can bound the total running time by $O(9^w poly(n))$. This can probably be improved to $O(3^w poly(n))$ using a similar approach as for the tables $S_j(\bar{U})$.

7. Practical Experiments and Conclusion

We started to implement the approach presented in this article in C++. The current version works for graphs over the vocabulary $\tau_{\mathrm{Graph}} = (\mathit{adj})$. At certain places, the implementation varies from the algorithms presented in this paper for increased efficiency. For instance, *reduce*() is usually not called explicitly but computed directly where needed.

We list some running times and memory usage of the implementation when solving the three problems discussed in the previous section. Input graphs are randomly generated subgraphs of $n \times m$ grids and Erdős–Rényi random graphs.

Minimum Vertex Cover

		time in seconds			memory in MB		
dimension	runs	min	max	median	min	max	median
1×200	40	0.2	0.2	0.2	1	1	1
2×100	40	0.3	0.4	0.3	1	1	1
3×66	40	0.5	0.9	0.6	1	1	1
4×50	40	0.1	1.0	0.1	1	1	1
5×40	40	0.2	0.4	0.3	1	2	2
6×33	40	0.3	0.9	0.5	2	3	2
7×28	40	0.6	1.9	1.0	2	5	3
8×25	40	1.2	4.6	2.3	3	9	5
9×22	40	1.9	13.6	5.2	5	18	10
10×20	40	4.4	41.4	13.7	9	36	19.5
11×18	40	11.3	156.4	46.2	16	62	39
12×16	40	28.2	642.4	185.2	27	128	76
13×15	40	61.3	2644.9	679.4	42	268	145.5
14×14	40	308.7	10257.3	3017.1	80	468	283

Minimum Dominating Set

		time in seconds			memory in MB		
dimension	runs	min	max	median	min	max	median
1×200	40	0.3	0.3	0.3	1	1	1
2×100	40	0.8	1.0	0.9	1	1	1
3×66	40	0.2	0.3	0.2	1	1	1
4×50	40	0.6	0.9	0.8	2	3	2.5
5×40	40	2.2	3.2	2.8	4	6	6
6×33	40	8.3	12.8	11.6	11	17	15
7×28	40	40.3	85.4	71.0	27	47	42
8×25	40	238.9	681.2	493.7	68	137	112
9×22	35	1605.7	8588.2	5235.3	170	386	332

3-Colorability

		time in seconds			memory in MB		
dimension	runs	min	max	median	min	max	median
1×200	20	0.5	0.6	0.5	1	1	1
2×100	20	0.2	0.2	0.2	1	1	1
3×66	20	0.7	1.6	0.9	2	2	2
4×50	20	3.3	6.3	4.8	5	5	5
5×40	20	15.3	38.3	29.3	10	15	14
6×33	20	99.6	317.4	233.0	26	45	42
7×28	20	771.6	2702.9	2262.0	70	139	123
8×25	15	4029.2	26841.6	14032.5	146	373	268

Table 1: Running times and memory usage on random subgraphs of grids with about 200 vertices

Minimum Vertex Cover

		time in seconds			memory in MB		
width	runs	min	max	median	min	max	median
1	387	0.5	0.8	0.6	2	3	2
2	179	0.1	1.0	0.8	2	4	3
3	68	0.1	0.3	0.2	3	4	3
4	74	0.2	0.5	0.3	3	4	4
5	69	0.4	1.3	0.7	3	4	4
6	62	0.9	2.3	1.4	4	6	5
7	38	1.5	5.5	3.1	5	11	9
8	36	2.5	14.0	6.3	8	20	15
9	45	7.4	34.9	16.6	18	40	27
10	29	24.8	121.6	56.8	30	78	56
11	28	55.8	382.1	156.8	56	138	103
12	29	164.6	1495.9	392.7	100	293	160

Minimum Dominating Set

		time in seconds			memory in MB		
width	runs	min	max	median	min	max	median
1	387	0.6	0.9	0.7	2	3	2
2	174	0.1	1.0	0.2	2	4	3
3	39	0.2	0.8	0.5	3	4	3
4	30	0.7	4.5	2.6	4	8	6
5	17	4.3	29.2	16.5	8	21	16
6	9	112.3	318.5	187.8	38	63	49
7	1	2403.9	2403.9	2403.9	162	162	162
8	3	35290.3	64922.8	51801.0	319	338	321
9	1	43228.4	43228.4	43228.4	347	347	347

3-Colorability

		time in seconds			memory in MB		
width	runs	min	max	median	min	max	median
1	387	0.1	0.2	0.1	2	3	2
2	174	0.2	0.8	0.5	2	4	3
3	39	0.8	3.6	2.2	3	6	5
4	30	4.1	22.2	15.0	7	16	13
5	17	35.3	156.9	99.9	19	52	37
6	9	485.8	1328.8	1168.8	81	150	125
8	2	33733.8	75099.9	54416.8	446	664	555

Table 2: Running times and memory usage for some random graphs on 200 vertices, grouped by the width of the tree decomposition used.

All graphs have about 200 vertices and the probability to include an edge ranges between 0.001 and 0.015. For the grid-subgraphs we used path decompositions of width n. Tree decompositions for the random graphs were computed by a triangulation heuristics (cf. [50]). The tests were done under Linux 2.6.32 on a Intel Core 2 Quad CPU Q6600 (2.40GHz) with 4 GB RAM.

8. Conclusion

Motivated by a practical application, we present an alternative proof of Courcelle's Theorem. Our proof is based on model checking games and tries to avoid expensive constructions such as the power set construction for tree automata, which turned out to cause some problems in practice.

Let us mention that our approach could be made simpler if we applied it to graphs of bounded clique-width. The union operation for *join* nodes of a tree decomposition involves a "fusion" of elements and of interpretations of nullary symbols. The clique-width parse trees do not use nullary symbols and the union is replaced by a disjoint union, which simplifies many of the operations. On the other hand, the lack of suitable algorithms to compute the mandatory clique-width parse trees favors treewidth based techniques for practical applications.

First experiments with our approach do indeed indicate practical feasibility. An implementation based on our proof can solve the 3-Colorability problem for some graphs where the automata theoretic approach based on the well-known MONA tool failed. The running times of our generic implementation can still not compete with specialized, hand-written algorithms that can easily solve problems such as, say 3-Colorability, for graphs of treewidth 15 and beyond. We are confident that further optimization can improve the feasibility of our generic approach in practical applications even more.

9. Acknowledgments

The authors thank an anonymous referee for valuable comments and suggestions that significantly helped to improve the quality of the paper. The authors thank Somnath Sikdar for useful discussions on earlier drafts of this paper. The third author thanks Bruno Courcelle for pointing out how to use incidence graphs to handle edge set quantifications in a simple way.

References

[1] B. Courcelle, The monadic second order theory of Graphs I: Recognisable sets of finite graphs, Information and Computation 85 (1990) 12–75.

[2] M. Frick, M. Grohe, The complexity of first-order and monadic second-order logic revisited, Ann. Pure Appl. Logic 130 (1–3) (2004) 3–31.

[3] S. Arnborg, J. Lagergren, D. Seese, Easy problems for tree-decomposable graphs, J. Algorithms 12 (2) (1991) 308–340.

[4] B. Courcelle, M. Mosbah, Monadic second-order evaluations on tree-decomposable graphs, Theor. Comput. Sci. 109 (1-2) (1993) 49–82.

[5] B. Courcelle, Graph rewriting: An algebraic and logic approach, in: Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B), Elsevier, 1990, pp. 193–242.

[6] R. G. Downey, M. R. Fellows, Parameterized Complexity, Springer-Verlag, 1999.

[7] J. Flum, M. Frick, M. Grohe, Query evaluation via tree-decompositions, J. ACM 49 (6) (2002) 716–752.

[8] J. Flum, M. Grohe, Parameterized Complexity Theory, Springer-Verlag, 2006.

[9] M. Weyer, Modifizierte parametrische Komplexitätstheorie, Ph.D. thesis, Universität Freiburg (2008).

[10] B. Courcelle, Graph Structure and Monadic Second-Order Logic, a Language Theoretic Approach, Cambridge University Press, 2011, book in preparation.

[11] S. Feferman, R. Vaught, The first order properties of algebraic systems, Fund. Math 47 (1959) 57–103.

[12] Y. Gurevich, Modest Theory of Short Chains. I, J. Symb. Log. 44 (4) (1979) 481–490.

[13] M. Frick, Easy Instances for Model Checking, Ph.D. thesis, Universität Freiburg (2001).

[14] J. A. Makowsky, Algorithmic uses of the Feferman-Vaught Theorem, Ann. Pure Appl. Logic 126 (1-3) (2004) 159–213.

[15] M. Grohe, Logic, graphs, and algorithms, in: J. Flum, E. Grädel, T. Wilke (Eds.), Logic and Automata: History and Perspectives, Amsterdam University Press, Amsterdam, 2007, pp. 357–422.

[16] N. Klarlund, A. Møller, MONA Version 1.4 User Manual, BRICS, Dept. of Comp. Sc., University of Aarhus, available from `http://www.brics.dk/mona/`. (January 2001).

[17] N. Klarlund, A. Møller, M. I. Schwartzbach, MONA Implementation Secrets, in: Proc. of CIAA00, Springer-Verlag, 2001, pp. 182–194.

[18] D. Soguet, Génération automatique d'algorithmes linéaires, Doctoral dissertation, University Paris-Sud (2008).

[19] B. Courcelle, J. A. Makowsky, U. Rotics, Linear Time Solvable Optimization Problems on Graphs of Bounded Clique Width, Theory of Computing Systems 33 (2000) 125–150.

[20] B. Courcelle, Graph structure and monadic second-order logic: Language theoretical aspects, in: Proceedings of the 35th International Colloquium on Automata, Languages, and Programming (ICALP), Vol. 5125 of Lecture Notes in Computer Science, Springer, 2008, pp. 1–13.

[21] G. Gottlob, R. Pichler, F. Wei, Abduction with bounded treewidth: From theoretical tractability to practically efficient computation, in: Proc. of 23rd AAAI, AAAI Press, 2008, pp. 1541–1546.

[22] G. Gottlob, R. Pichler, F. Wei, Monadic datalog over finite structures of bounded treewidth, ACM Trans. Comput. Logic 12 (1) (2010) 3:1–3:48.

[23] B. Courcelle, I. A. Durand, Verifying monadic second-order graph properties with tree automata, in: 3rd European Lisp Symposium, 2010, pp. 7–21, informal proceedings edited by C. Rhodes.

[24] B. Courcelle, I. A. Durand, Tractable constructions of finite automata from monadic second-order formulas, presented at *Logical Approaches to Barriers in Computing and Complexity*, Greifswald, Germany (2010).

[25] B. Courcelle, Special tree-width and the verification of monadic second-order graph properties, in: Proceedings of the 30th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS), Vol. 8 of Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2010, pp. 13–29.

[26] J. Hintikka, Logic, Language-Games and Information: Kantian Themes in the Philosophy of Logic, Clarendon Press, 1973.

[27] E. Grädel, Finite model theory and descriptive complexity, in: Finite Model Theory and Its Applications, Springer, 2007, pp. 125–230.

[28] E. Grädel, Back and forth between logics and games, in: K. R. Apt, E. Grädel (Eds.), Lectures in Game Theory for Computer Scientists, Cambridge University Press, 2011, pp. 99–145.

[29] K. A. Abrahamson, M. R. Fellows, Finite automata, bounded treewidth, and well-quasiordering, in: Graph Structure Theory, Contemporary Mathematics, Vol. 147, American Mathematical Society, 1993, pp. 539–564.

[30] J. W. Thatcher, J. B. Wright, Generalized finite automata theory with an application to a decision problem of second-order logic, Mathematical Systems Theory 2 (1) (1968) 57–81.

[31] J. Doner, Tree acceptors and some of their applications, J. Comput. Syst. Sci. 4 (1970) 406–451.

[32] P. B. Miltersen, J. Radhakrishnan, I. Wegener, On converting cnf to dnf, Theor. Comput. Sci. 347 (1-2) (2005) 325–335.

[33] B. Courcelle, On the model-checking of monadic second-order formulas with edge set quantifications, Discrete Applied Mathematics, to appear.

[34] K. L. McMillan, A technique of state space search based on unfolding, Form. Methods Syst. Des. 6 (1995) 45–65.

[35] H.-D. Ebbinghaus, J. Flum, Finite Model Theory, Springer, 1999.

[36] R. Ganian, P. Hliněený, On parse trees and Myhill–Nerode–type tools for handling graphs of bounded rank-width, Disc. App. Math. 158 (7) (2010) 851–867.

[37] N. Robertson, P. D. Seymour, Graph minors. II. Algorithmic aspects of tree-width, J. Algorithms 7 (1986) 309–322.

[38] R. Diestel, Graph Theory, 4th Edition, Springer-Verlag, Heidelberg, 2010.

[39] S. Arnborg, D. G. Corneil, A. Proskurowski, Complexity of finding embeddings in a k-tree, SIAM J. Alg. Disc. Meth. 8 (1987) 277–284.

[40] H. L. Bodlaender, A linear time algorithm for finding tree-decompositions of small treewidth, SIAM J. Comput. 25 (1996) 1305–1317.

[41] H. Bodlaender, A. M. C. A. Koster, Treewidth computations I. Upper bounds, Inf. Comput. 208 (3) (2010) 259–275.

[42] H. L. Bodlaender, A tourist guide through treewidth, Acta Cybernetica 11 (1993) 1–21.

[43] H. L. Bodlaender, A partial k-arboretum of graphs with bounded treewidth, Theoretical Comput. Sci. 209 (1998) 1–45.

[44] A. Tarski, The semantic conception of truth, Philosophy and Phenomenlological Research 4 (1944) 13–47.

[45] C. Baier, J.-P. Katoen, Principles of Model Checking (Representation and Mind Series), The MIT Press, 2008.

[46] J. A. Telle, A. Proskurowski, Algorithms for vertex partitioning problems on partial k-trees, SIAM Journal on Discrete Mathematics10 (4) (1997) 529–550.

[47] J. M. M. van Rooij, H. L. Bodlaender, P. Rossmanith, Dynamic programming on tree decompositions using generalised fast subset convolution, in: ESA, Vol. 5757 of Lecture Notes in Computer Science, Springer, 2009, pp. 566–577.

[48] D. Lokshtanov, D. Marx, S. Saurabh, Known algorithms on graphs of bounded treewidth are probably optimal, Tech. Rep. abs/1007.5450, CoRR, accepted for SODA'11 (Jul. 2010).

[49] J. Alber, R. Niedermeier, Improved tree decomposition based algorithms for domination-like problems, in: Proceedings of the 5th Symposium on Latin American Theoretical Informatics (LATIN), no. 2286 in Lecture Notes in Computer Science, Springer, Cancun, Mexico, 2002, pp. 613–627.

[50] H. L. Bodlaender, Necessary edges in k-chordalisations of graphs, J. Comb. Optim. 7 (3) (2003) 283–290.